The first programming language and freshman year in computer science: characterization and tips for better decision making

Sónia Rolland Sobral [0000-0002-5041-3597]

[1] REMIT, Universidade Portucalense, Porto, Portugal
soniarollandsobral@gmail.com

Abstract. The ability to program is the "visible" competency to acquire in an introductory unit in computer science. However, before a student is able to write a program, he needs to understand the problem: before formalizing, the student must have to (be able) to think, (be able) to solve and (be able) to define. At an early stage of learning there are no significant differences between programming languages.

The discussion of the early programming language continues: probably never will be a consensus among academics. The Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE) computer science curriculum recommendations haven't clearly defined which programming language to adopt: it is the course directors and teachers who must make this choice, consciously and not only following the trends.

This article presents a set of items that should be considered when you make a programming language choice for the first programming unit in higher education computer science courses.

Keywords: Programming Languages, Undergraduate Studies; Introduction to Programming.

1 Introduction

Programmability is the "visible" skill to be acquired in an introductory unit in computer science. Programming can be considered an art [1], a science [2], a discipline [3] or even the science of abstraction [4]. However, using a programming language is no more than a method for the programmer can communicate instructions to the computer.

Before designing a program, the student must know the problem, know how to use the necessary tools to solve the problem that need to be solved with the machine, such as methods used to specify specifications and rigorous solutions that can be implemented on the computer. For this, the student will also have to learn one or more programming languages and paradigms in order to use the programming notions and systematize the use of data structures and algorithms to solve different categories of problems [5].

Students often have the perception that the focus is on learning the syntax of the programming language, leading them to focus on implementation activities rather than activities such as planning, drawing, or testing [6].

The art of programming involves four steps [7]:

a) To Think: the conceptualization and analysis phase in which problems are divided into small and easily intelligible processes or tasks, the modular structure, whose organization must follow a descending programming logic, Top-Down [8];

b) To Solve: Translate Top-Down into Algorithm [1], which incorporates solution rules using pseudo code;

c) To Define: Using variables and data structures, characterize the data model to be used in the algorithm;

d) To Formalize: translate the algorithm into a programming language, its implementation and execution on the computer.

Then it comes the most important phase, the true moment of truth: Does the program run, is error-free and give the correct result? And how are you sure that the result is "the" or "probably" the correct solution?

The following table shows how each of ten of the most well-known programming languages write the famous "Hello, World!"

Table 1. "Hello World!" ten different programming languages.

Prog. Language	Write Hello World
C	printf ("'Hello World!");
C#	Console.WriteLine("Hello World!");
C++	cout<<"Hello World"
COBOL	display "Hello world!".
Fortran	print *,"Hello world!"
Java	System.out.println("'Hello World!");
JavaScript	document.write("Hello world!");
Pascal	writeln ('Hello World!');
Python	print "Hello, World!"
Visual Basic.NET	Console.WriteLine("Hello world!")

Each of the ten programming languages presented in the previous table has a different notation, however it's quite similar in a basic algorithm like "Hello World!".

Some say that programming is very difficult [9] [10] while for others it may be easy [11].

Success is achieved through a good deal of study, research, planning, persistence and preferably a passion for the activity.

This article is divided into five parts: this introduction, the second part with Programming languages: concept and characterization; the third part with Evolution of programming languages in undergraduate computer science studies; the fourth part with Choosing the Initial Programming Language and the last part with conclusions and future work.

2. Programming languages: concept and characterization

A programming language is a system that allows the interaction between man and the machine, being "understood" by both. It is a formal language that specifies a set of instructions and rules. Programming languages are the medium of expression in the art of computer programming. Program writing must be succinct and clear, because programs are meant to be included, modified, and maintained throughout life: a good programming language should help others to read programs and to understand how they work. [12]. A program is a set of instructions that make up a solution after being coded in a programming language. [13].

There are several reasons why thousands of high-level programming languages exist and new ones continue to emerge [14]:

Evolution: The late 1960s and early 1970s saw a revolution in "structured programming," in which the GoTo-based flow control of languages such as FORTRAN, COBOL, and Basic gave way to while loops, case statements (switch). In the late 1980s, Algol, Pascal and Ada began to give way to the object-oriented languages like Smalltalk, C ++ and Eiffel. And so on.

- Special Purposes: Some programming languages are designed for specific purposes. C is good for low level system programming. Prolog is good for reasoning about logical relationships between data. Each can be successfully used for a wide range of tasks, but the emphasis is clearly on the specialty.

- Personal preference: Different people like different things. Some people love C while others hate it, for example.

According to Stack Overflow Annual Developer Survey [15], with over 90,000 answers to over 170 countries, by 2019 the most widely used programming language is JavaScript (Table).

***Table* 2.** Top15, Programming languages most used in 2019 [15].

PL	%
JavaScript	67.8%
HTML/CSS	63.5%
SQL	54.4%
Python	41.7%
Java	41.1%
Bash/Shell/PowerShell	36.6%
C#	31.0%
PHP	26.4%
C++	23.5%

TypeScript	21.2%
C	20.6%
Ruby	8.4%
Go	8.2%
Assembly	6.7%
Swift	6.6%

In September 2019, the TIOBE Programming Community index [16],

Indicator of the popularity of programming languages featured Java as the most popular (Table 3), followed by C and Python.

Table 3. Top7, Indicator of popularity of programming languages, TIOBE [16].

LP	Ratings
Java	16,66%
C	15,21%
Python	9,87%
C++	5,64%
C#	3,40%
Visual Basic .NET	3,29%
JavaScript	2,13%

The technology of the most searched electronic sites (Table 4) according to Wikipedia[1] (Wikipedia, 2019) is also varied in the use of back-end languages; However, JavaScript is almost always used on the front end.

Table 4. The technology used in the most searched websites [17].

Web Site	Back-end Language
Amazon.com	Java, C++, Perl
Bing	C++, C#
eBay.com	Java, JavaScript, Scala
Facebook.com	Hack, PHP (HHVM), Python, C++, Java, Erlang, D, XHP, Haskell
Google.com	C, C++, Go, Java, Python
Linkedin.com	Java, JavaScript, Scala
MSN.com	C#
Pinterest	Python (Django), Erlang
Twitter.com	C++, Java, Scala, Ruby
Wikipedia.org	PHP, Hack
WordPress.com	PHP
Yahoo	PHP
YouTube.com	C, C++, Python, Java, Go

3. Evolution of programming languages in undergraduate computer science studies

Computer science became a recognized academic field in October 1962 with the creation of Purdue University's first department. [18]. The first curriculum studies appeared in March 1968, when the Association for Computing Machinery (ACM) published an innovative and necessary document, Curriculum 68: Recommendations for academic programs in computer science [19], with early indications of curriculum models for programs in computer science and computer engineering. Prerequisites, descriptions, detailed

[1] Wikipedia is not a reliable source of information because it has collaborative features!

sketches, and annotated bibliographies were included for each of these courses. As initial unit, it presented B1. Introduction to Computing (2-2-3)[2] in which an algorithmic language was proposed, recommending that only one language be used or two "in order to demonstrate the wide diversity of the computer languages available"; "Because of its elegance and novelty, SNOBOL can be used quite effectively for this purpose."

With the emergence of many new courses and departments, ACM published a new report, Curriculum'78: recommendations for the undergraduate program in computer science [20], updating Curriculum 68. It presented for the first time the denomination CS1: Computer Programming I (2-2-3): "The emphasis of the course is on the techniques of algorithm development and programming with style. Neither esoteric features of a programming language nor other aspects of computers should be allowed to interfere with that purpose."

Despite the importance of Curriculum'78 there has been much discussion, particularly regarding the sequence CS1 and CS2. In 1984 a new report is published: "Recommended curriculum for CS1, 1984" [21] to detail a first computer science course that emphasizes programming methodology and problem solving." This report refers Pascal, PL / 1 e Ada: "These features are important for many reasons. For example, a student cannot reasonably practice procedural and data abstraction without using a programming language that supports a wide variety of structured control features and data structures". They said that "Although FORTRAN and BASIC are widely used, we do not regard either of these languages as suitable for CS1" and ALGOL "does satisfy the requirements but is omitted from our list of recommended languages simply because it is no longer widely used or supported."

In 1991 [22] IEEE (Institute of Electrical and Electronics Engineers) and ACM joined for a new document. This document emerged by breaking with some of the concepts of previous documents, presenting a set of individual knowledge units corresponding to a topic that should be addressed at some point in the undergraduate program. In this way, institutions have considerable flexibility in setting up course structures that meet their particular needs.

In 2001 a new document was published [23]. This document questioned the programming-first of previous documents, as early programming approaches may lead students to believe that writing a program is the only viable approach to solving problems using a computer and that focus only on programming reinforces the common misperception that "computer science" equals programming. They said "In fact, the problems of the programming-first approach can be exacerbated in the objects-first model because many of the languages used for object-oriented programming in industry—particularly C++, but to a certain extent Java as well—are significantly more complex than classical languages. Unless instructors take special care to introduce the material in a way that limits this complexity, such details can easily overwhelm introductory students".

In 2008 a new report is presented: "Computer Science Curriculum 2008: An Interim Revision of CS 2001" [24]; security is strongly mentioned, making minor revisions to the 2001 document. Curriculum'2008 reinforces the idea that "Computer science professionals frequently use different programming languages for different purposes and must be able to learn new languages over their careers as the field evolves. As a result, students must recognize the benefits of learning and applying new programming languages. It is also important for students to recognize that the choice of programming paradigm can significantly influence the way one thinks about problems and expresses solutions of these problems. To this end, we believe that all students must learn to program in more than one paradigm".

When referring to languages and paradigms, the "Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science" [25], says that the choice of programming languages seems to be depend on the chosen paradigm and "There does, however, appear to be a growing trend toward "safer" or more managed languages (for example, moving from C to Java) as well as the use of more dynamic languages, such as Python or JavaScript." "Visual programming languages, such as Alice and Scratch, have also become popular choices to provide a "syntax-light" introduction to programming; these are often (although not exclusively) used with non-majors or at the start of an introductory course". And "some introductory course sequences choose to provide a presentation of alternative programming paradigms, such as scripting vs. procedural programming or functional vs. object-oriented programming, to give students a greater appreciation of the diverse perspectives in programming, to avoid language-feature fixation, and to disabuse them of the notion that there is a single "correct" or "best" programming language".

[2] (2-2-3) two hours of lectures and two hours of laboratory per week for a total of three semester hours of credit.

It is clear that curriculum recommendations do not indicate which programming language to adopt. However, it is always said that they should have the simplest possible usability and syntax for better learning. Language choice has always been a matter of concern to educators [26] [27] [28] [29] [30].

FORTRAN was selected as a high level language for the first introductory courses; especially those linked to engineering departments. The less widely used COBOL was adopted by departments that were more closely linked to information systems. [31]. At that time you couldn't talk about methodology: everything was just programming.

With the emergence of BASIC in 1964 [32] has led some departments to use this language for introductory students. In 1972 almost all computer science degree programs used ALGOL, FORTRAN or LISP, while most data processing programs used COBOL. In Britain, BASIC was also important. In the late 60's, some departments tried various languages like PL / I [33].

With Dijkstra's manifest [34] structured programming begins to be discussed [35] [36]. With the emergence of the Pascal language [37] seems to become almost consensual [31]: an almost written language for the purpose of programming learning, using a very friendly development environment [38], and obviously because of the proliferation of personal computers and the availability of Pascal compilers [39].

Pascal's decline began in the late 1980s, early 1990s, with object-oriented programming. And also because Pascal has a difficult document reuse, but also because Pascal is not a "real world" language [39]. McCauley e Manaris [40]

They say that as a first language Pascal was used by 36% and C ++ by 32% in 1995-1996 but 22% intended to make a switch to C ++, C, Ada or Java. There are several studies that present the evolution of the languages adopted in initial programming curricular units [41] [42] and even lists of programming languages taught in various courses [43].

In Portugal [44], in the 2016-2017 school year, the most common first-year programming language sequence in 46 courses analyzed was C (48%), followed by Java (22%), C and Haskell (9%), C and Java (4%), Scheme and Java (4%). There were also residual sequences Excel and C, Python, Python, HTML and Java, Python and Java, Schem and C ++ and XML and Java. Regarding the ten Portuguese first cycle (or with integrated master's degree) courses in Computer Engineering considered most significant [45], it was found that the most common sequences were only Java or Python and C (both with 30%), C (20%), Python and Java or Haskell and C (both with 10%).

According to the document "An Analysis of Introductory Programming Courses at UK Universities" [46]:

- 73.8% use only one programming language; 21% reported using two.
- The most widely used language is Java (46%), followed by the "C family" (C, C ++ and C #) (23.6%) and Python (13.2%). Javascript and Haskell are much less adopted.
- The reason given by 82.7% of those who uses Java, was to be object oriented, while 72.7% of those using Python refer to the pedagogical benefits.

According to the document "Introductory Programming Courses in Australasia in 2016" [46] referring to the Universities of Australia and New Zealand:

- 48 courses studied: 15 used Java, 15 Python, 8 C, 5 C #, 2 Visual Basic and 2 Processing. The remaining ten use another programming language.
- The reasons given for choosing Python and Java are quite different: pedagogical benefits for Python (67%), availability / cost (53%) and platform independence (40%). The reasons given for using Java are industry-relevant (92%), object-oriented (86%), and platform independence (62%).

"What language? - The choice of an introductory programming language" [47], A study with 496 four-year courses in the United States, refere that Java is used by 41.94%, Python 26.45%, C ++ 19.35%, C 4.52%, C # 0.65% and 7.10% by another. The reasons for choosing were: Programming language features 26.19%, Ease of learning 18.81%, Job opportunities for students 14.76%, Popularity at the academy 13.10%, Institutional tradition 8.57%, choice of advisory board 5.95%, availability of teachers or scheduling restrictions 5%.

A 2016 study [48] analyse 218 colleges and 143 universities in 35 European countries, indicating that the most commonly used programming language was C (30.6%), following C ++ (21.9%) and Java (20.7%)

A document [10] for 152 CS1 units from a number of different countries concludes that Java is by far the most common CS1 language, used in 74 (49%) of the 152 programs. The second most frequent is Python, with 36 (24%). C ++ comes in 30 (20%) followed by C in 8 (5%), with the most obvious change being the rise of Python which "probably occurred at the expense of Java and C++".

Today, with few exceptions, the academy follows the "real world" and the "C family" (C, C ++, C #), Python, Java, and JavaScript are undoubtedly the programming languages adopted in introductory programming units.

4. Choosing the Initial Programming Language

In 2004, Eric Roberts [49] commented that the languages, paradigms, and tools used to teach computer science became increasingly complex; which pressures to cover more material in an already overcrowded area. The problem of complexity is exacerbated by the fact that languages and tools change rapidly, leading to profound instability in the way computer science is taught. Roberts predicted that Java would be the way "we must take responsibility for breaking this cycle of rapid obsolescence by developing a stable and effective collection of Java-based learning applications that meet the needs of the science education community".

Dijkstra [50] wrote about the importance of the chosen programming language: "the tools we are trying to use and the language or notation we are using to express or record our thoughts, are the major factors determining what we can think or express at all! The analysis of the influence that programming languages have on the thinking habits of its users, and the recognition that, by now, brainpower is by far our scarcest resource, they together give us a new collection of yardsticks for comparing the relative merits of various programming languages. "

When selecting the first programming language for introductory programming courses, it is important to consider whether it is suitable for teaching and learning. Over time various pseudo-code languages have been created in search of the perfect teaching language but no definitive solution has been found [51].

In document "Introductory Programming Subject in European Higher Education" [48] discusses the need to teach introductory programming using educational programming languages. But in the past these languages have been discontinued: the Pascal language being the most visible.

The programming language chosen for introductory programming courses often seems like a religious or football issue. In reflection-teaser "The Programming Language Wars" [52] it is even said that "Programming language wars are a major social problem causing serious problems in our discipline" leading to "massively duplicating efforts" and "reinventing the wheel constantly." Choosing the best programming language is often an emotional issue, leading to major debates [53] but for Guerreiro [54] "It is up to us to have an open, exploratory attitude and at the same time not dogmatically accept what those who make the most noise say. In fact, I think we should even pass this on to students too, to help them develop their critical thinking, and to be able, sooner or later, to choose the languages and tools that can best respond to their needs".

In fact, two of the most important points are pedagogical issues and student preparation for the world of work. Parker e Devey [33] define them as pragmatic and pedagogical: industry acceptance, market penetration as well as the employability of graduates.

Keep in mind that "small programming" needs to be mastered before "large programming" [55] since traditionally only "in the third or fourth year are faced with the problems that arise in the design of large programs." Collberg [55] said that the task of choosing the initial language is not an easy task. It must obey factors such as simplicity, expressiveness, suitability for tasks, availability of accessible resources, and reliable compilers.

Programming languages are the fundamental basis of programming, but trends change dramatically over time. Professionals will not use the same programming language, or even the same programming model, for their entire professional career. In addition, well-informed language choices can make a huge difference in programmer productivity and program quality. Therefore, it is crucial that students master the essential concepts of programming languages, so that they can choose and use languages based on a deep understanding of the abstractions they express and their ability to solve programming problems. [56].

Choosing the initial programming language to adopt should take into account several points: Course objectives, Teacher preferences, available implementations, and relationships with other course units, as well as the "real world": students are often more motivated to study a familiar language that is known to be requested by employers [57].

Howatt [58] uses an evaluation method for programming languages using several items: language design and implementation (accuracy and speed), human factors (usability and ease), software engineering (portability, reliability and reuse) and application mastery specific applications).

The paradigm chosen can be very important. [59] unless one adopts "exposing students to all major paradigms through the use of a multiparadigmatic language, and does not attempt to identify" the "correct paradigm" [60].

The document “A Formal Language Selection Process” [61] has a design of choice with a weighted multicriteria method and where evaluation criteria are identified such as Reasonable Financial Cost, Academic / Student Version Availability, Academic Acceptance, Textbook Availability, Lifecycle Stadium, Industry Acceptance, Marketing (regional and national), Student / Academic / Full System Requirements, Operating System Dependency, Proprietary / Open Source, Development Environment, Debugging Facilities, Fundamentals Learning Ease, Secure Code, Advanced Course Features Subsequent, More or Less Complicated Programming, Web Development Support, Teaching Support, Object Oriented Support, Support Availability, Instructor and Staff Teaching, and Expected Level of New Students.

Mannilla e de Raadt [30] compare multiple languages in which various inclusion / exclusion criteria are used such as Be suitable for teaching, Be interactive and fast, Promotes correct writing, Allows you to program in "small", Provides a continuous development environment, Good user community, Open source, good support, be free, have good teaching material, not only be used for educational purposes only, be reliable and efficient.

Several attempts have been made in the past to sort programming languages [41].

There are numerous comparisons between the most commonly used languages: like Python vs C ++ [62], Python vs C [63], Java vs Python [64], C++ vs. Java [65]. Any of the three / four most commonly used programming languages is free, well supported and has a large user community, is reliable and efficient.

Ease of learning can be discussed: C will have a more complicated syntax than Pyhton. The major differences are the use of pointers (C only), parameter passing by reference and value (C only), programming paradigm (procedural in C, object oriented in others), being compiled or interpreted (C and Pyhton / Java respectively).

5. Conclusions

A programming language is used to materialize the solution of a problem. A program should only be written after finding the best solution.

There are numerous programming languages that are adopted for the sake of evolution, purpose of use or even personal taste.

The choice of which programming language to choose for introductory teaching must accompany evolution, but because it has a propaedeutic character, the choice must meet several requirements, namely pedagogical, and acceptance from the outside world.

As future work we will compare the three programming languages currently used for CS1 curricular units: compare the simplicity, the IDE, the debugger and other features that have been identified in this article.

There isn’t, and probably never will be, consensus as to which language should be chosen to introduce the student in the world of computer science.

The first programming language of a future computer science professional is just the beginning of a long walk.

References

[1] D. Knuth, The Art of Computer Programming, Addison-Wesley, 1968.

[2] D. Gries, The science of programming, Springer, 1981.

[3] E. W. Dijkstra, A Discipline of Programming, Prentice Hall, 1976.

[4] A. Aho and J. D. Ullman, Foundations of Computer Science: C Edition (Principles of Computer Science Series), W. H. Freeman, 1994.

[5] S. R. Sobral, B-learning em disciplinas introdutórias de programação, Guimarães: Universidade do Minho, 2008.

[6] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," in *ITiCSE on Innovation and technology in computer science education*, 2001.

[7] S. R. Sobral and P. Pimenta, "O ensino da programação: exercitar a distancia para combate às dificuldades," in *4ª Conferência Ibérica de Sistemas e Tecnologias de Informação*, 2009.

[8] J. R. Lima, Programação de computadores, Porto Editora, 1991.

[9] S. Bergin and R. Reilly, "Programming: Factors that Influence SuccessSusan," in *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 2005.

[10] B. A. Becker and T. Fitzpatrick, "What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?," in *50th ACM Technical Symposium on Computer Science Education*, 2019.

[11] A. Luxton-Reilly, "Learning to Program is Easy," in *ACM Conference on Innovation and Technology in Computer Science Education*, 2016.

[12] J. C. Mitchell, Concepts in programming languages, Cambridge University Press, 2003.

[13] M. Sprankle, Problem Solving and Programming Concepts, 9 edition ed., Pearson, 2011.

[14] M. L. Scott, Programming Language Pragmatics, 3rd edition ed., Elsevier, 2009.

[15] Stackoverflow.com, "Stackoverflow," 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019.

[16] TIOBE Software BV, "TIOBE," Set. 2019. [Online]. Available: https://www.tiobe.com/tiobe-index/.

[17] Wikipedia, "Programming languages used in most popular websites," Setembro 2019. [Online]. Available: https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites.

[18] J. R. Rice and S. Rosen, "History of the Computer Sciences Department at Purdue University," Department of Computer Science, Purdue University, 1990.

[19] W. F. Atchison, S. D. Conte , J. W. Hamblen , T. E. Hull , T. A. Keenan , W. B. Kehl , E. J. McCluskey , S. O. Navarro , W. C. Rheinboldt, E. J. Schweppe, W. Viavant and D.

M. Young, Jr., "Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science," *Communications of the ACM*, vol. v.11 n.3, pp. 151-197, Março 1968.

[20] R. H. Austing, B. H. Barnes, D. T. Bonnette, G. L. Engel and G. Stokes, "Curriculum '78: recommendations for the undergraduate program in computer science— a report of the ACM curriculum committee on computer science," *Communications of the ACM*, vol. v.22 n.3, pp. 147-166, Março 1979.

[21] E. B. Koffman , P. L. Miller and C. E. Wardle, "Recommended curriculum for CS1, 1984," *Communications of the ACM*, vol. v.27 n.10, pp. .998-1001, Outubro 1984.

[22] A. B. Tucker and ACM/IEEE-CS Joint Curriculum Task Force., Computing curricula 1991 : report of the ACM/IEEE-CS Joint Curriculum Task Force, ACM Press, 1990, p. 154.

[23] The Joint Task Force IEEE and ACM, "CC2001 Computer Science, Final Report," 2001.

[24] L. Cassel, A. Clements, G. Davies, M. Guzdial and R. McCauley, "Computer Science Curriculum 2008: An Interim Revision of CS 2001," ACM, 2008.

[25] Task force ACM e IEEE, "Computer Science Curricula 2013," ACM and the IEEE Computer Society, 2013.

[26] C. Smith and J. Rickman, "Selecting Languages for Pedagogical Tools in the Computer Science Curriculum," in *Proceedings of the sixth SIGCSE technical symposium on Computer science education*, 1976.

[27] R. L. Wexelblat, "Discussion), First programming language: Consequences (Panel," in *Discussion), First programming language: Consequences (Panel*, 1979.

[28] A. L. Tharp, "Selecting the "right" programming language," in *SIGCSE '82 technical symposium on Computer science education*, Indianapolis, Indiana, USA, 1982.

[29] R. Duke, E. Salzman, J. Burmeister, J. Poon and L. Murray, "Teaching programming to beginners - choosing the language is just the first step," in *ACSE '00 Proceedings of the Australasian conference on Computing education*, 2000.

[30] L. Mannila and M. d. Raadt, "An objective comparison of languages for teaching introductory programming," in *6th Baltic Sea conference on Computing education research: Koli Calling 2006*, 2006.

[31] E. Giangrande Jr., "CS1 programming language options," *Journal of Computing Sciences in Colleges*, vol. v22, no. 3, pp. 153-160, 2007.

[32] J. G. Kemeny and T. E. Kurtz, BASIC - A Manual for BASIC, the elementary algebraic language, Dartmouth College, 1964.

[33] K. Parker and B. Davey, "The History of Computer Language Selection," *IFIP Advances in Information and Communication Technology*, pp. 166-179, 2012.

[34] E. W. Dijkstra, "Go To Statement Considered Harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147-148, 1968.

[35] D. Knuth, "Structured Programming with go to Statements," *Computing Surveys*, vol. 6, no. 4, pp. 261-301, 1974.

[36] O. Dahl, E. Dijkstra and C. Hoare, Structured programming, Academic Press Ltd, 1972.

[37] N. Wirth, "The Programming Language Pascal," in *Pioneers and Their Contributions to Software Engineering*, Springer, 1971.

[38] D. Gupta, "What is a good first programming language?," *Crossroads, The ACM Magazine for Students*, vol. 10, no. 4, 2004.

[39] S. Levy, "Computer language usage in CS1: survey results," *ACM SIGCSE Bulletin*, vol. 7, no. 3, pp. 21-26, 1995.

[40] R. McCauley and B. Manaris, "Computer science degree programs: what do they look like? A report on the annual survey of accredited programs," *ACM SIGCSE Bulletin,* vol. 30, no. 1, pp. 15-19, 1998.

[41] K. S. A. F. I. S. A. A. Farooq MS, "An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages," *PLoS ONE ,* 2014.

[42] S. R. Sobral, "30 YEARS OF CS1: PROGRAMMING LANGUAGES EVOLUTION," in *12th annual International Conference of Education, Research and Innovation*, 2019.

[43] G. D. M. N. S. J. Siegfried RM, "A Longitudinal Analysis of the Reid List of First," *Information Systems Education Journal ,* vol. 10, no. 4, pp. 47-54, 2016.

[44] S. R. Sobral, "Bachelor's and master's degrees integrated in Portugal in the area of computing: a global vision with emphasis on programming UCS and programming languages used," in *11th annual International Conference of Education, Research and Innovation*, 2018.

[45] S. R. Sobral, "Introduction to programming: Portrait of Higher Education in computer science in Portugal," in *11th International Conference on Education and New Learning Technologies*, 2019.

[46] E. Murphy1, T. Crick2 and J. H. Davenport, "An Analysis of Introductory Programming Courses at UK Universities," *The Art, Science, and Engineering of Programming,* vol. 1, no. 2, 2017.

[47] O. Ezenwoye, "What language? - The choice of an introductory programming language," *48th Frontiers in Education Conference, FIE 2018,* 2018.

[48] V. ALEKSIĆ and M. IVANOVIĆ, "Introductory Programming Subject in European Higher Education," *Informatics in Education,* vol. 15, no. 2, p. 163–182, 2016.

[49] E. Roberts, "The Dream of a Common Language: The Search for Simplicity and Stability in Computer Science Education," in *35th SIGCSE technical symposium on Computer science education*, 2004.

[50] E. W. Dijkstra, "The Humble Programmer," *Communications of ACM*, vol. 15, no. 10, 1972.

[51] M. Laakso, E. Kaila, T. Rajala and T. Salakoski, "Define and Visualize Your First Programming Language," in *8th IEEE International Conference on Advanced Learning*, 2008.

[52] A. Stefik and S. Hanenberg, "The Programming Language Wars: Questions and Responsibilities for the Programming Language Community," in *2014 ACM International*

Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, 2014.

[53] L. Goosen, "A Brief History of Choosing First Programming languages," *History of Computing and Education 3*, 2008.

[54] P. Guerreiro, "A mesma velha questão: como ensinar Programação?," in *Quinto Congreso Iberoamericano de Educación Superior*, 1986.

[55] C. S. Collberg, "Data structures, algorithms, and software engineering," in *Software Engineering Education - SEI Conference 1989*, 1989.

[56] K. Bruce, S. N. Freund, R. Harper, J. Larus and G. Leavens , "What a Programming Languages Curriculum Should Include," in *SIGPLAN Workshop on Undergraduate Programming Language Curricula*, 2008.

[57] K. N. King, "The evolution of the programming languages course," *ACM SIGCSE Bulletin*, vol. 24, no. 1, pp. 213-219, 1992.

[58] J. Howatt, "A project-based approach to programming language evaluation," *ACM SIGPLAN Notices*, vol. 30, no. 7, pp. 37-40, 1995.

[59] P. A. Luker, "Never mind the language, what about the paradigm?," in *twentieth SIGCSE technical symposium on Computer science education*, 1989.

[60] T. A. Budd and R. K. Pandey, "Never Mind the Paradigm, What About Multiparadigm Languages?," *ACM SIGCSE Bulletin*, vol. 27, no. 2, pp. 25-30, 1995.

[61] K. R. Parker, J. T. Chao, T. A. Ottaway and J. Chang, "A Formal Language Selection Process," *Journal of Information Technology Education*, vol. 5, no. 1, pp. 133-151, 2006.

[62] 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 *49th ACM Technical Symposium on Computer Science Education*, 2018.

[63] J. Wainer and E. Xavier, "A Controlled Experiment on Python vs C for an Introductory Programming Course: Students' Outcomes," *ACM Transactions on Computing Education*, vol. 18, no. 3, 2018.

[64] K. McMaster, S. Sambasivam, R. Ragueand S. Wolthuis, "Java vs. Python Coverage of Introductory Programming Concepts: a Textbook Analysis," *Information Systems Education Journal*, vol. 15, no. 3, pp. 4-13, 2017.

[65] W. Farag, S. Ali and D. Deb, "Does language choice influence the effectiveness of online introductory programming courses?," in *14th annual ACM SIGITE conference on Information technology education*, 2013.

[66] E. B. Koffman, D. Stemple and C. E. Wardle, "Recommended curriculum for CS2, 1984," *Communications of the ACM*, vol. 28, no. 8, pp. 815-818, 1985.

[67] E. B. Koffman , D. Stemple and C. E. Wardle , "Recommended curriculum for CS2, 1984: a report of the ACM curriculum task force for CS2," *Communications of the ACM*, vol. 28, no. 8, pp. 815-818, 1985.

[68] The Joint Task Force for Computing Curricula 2005, "Computing Curricula 2005: The Overview Report," ACM, 2005.

[69] The Joint Task Force on Computing Curricula, "Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," ACM, 2004.

[70] The Joint Task Force on Computing Curricula , "SE2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," ACM, 2004.