30 YEARS OF CS1: PROGRAMMING LANGUAGES EVOLUTION

S. Sobral

Universidade Portucalense (PORTUGAL), REMIT

Abstract

CS1 (computer science 1) is a course that aims to introduce college students to a first contact with the computing world, especially simple computer-coded everyday problems. The lessons focus on computational thinking and introduce the use of one (or several) programming languages for students beginning their university course. CS1 plays an important role in the academic and professional life of new computer scientists.

For this reason, this course unit is very much associated with programming languages. Curricula are not much different, but programming languages have changed a lot over the years. How was this change? Are there clues to these changes? Are there logical reasons why the choice of programming language does not hold up?

What programming language should the teacher choose to teach students? The discussion about the early programming language is long: there have always been various trends and their fervent supporters, as well as those who are always critical.

This article makes a historical review of the programming languages used in the introduction of computing course over the last 30 years, differentiating an evolution of programming language choices in the past century and now. At the end of the article, an evolutionary trend is listed by articles published by people involved in the subject. The methodology used in this article compiles the Google Scholar articles for each of the last 30 years (1988-2018) and analyses which programming languages are used in academic studies. It is very interesting to see how the programming languages used in the introductory programming units change over the years.

Keywords: Programming languages, CS1.

1 INTRODUCTION

The importance of the role of the disciplines (or curricular units) introductory programming in the curriculum of a Computer Degree is unquestionable: their goal makes them key curricular units that become very important in the training of students and their performance, which has a decisive influence on the success of students in their academic and professional career [1].

With the course units of a propaedeutic nature, a student should, among other skills to achieve, be able to develop and implement computer solutions for problem solving, that is, learn correctly and effectively to program. Before designing a program, the student must understand the problem, develop strategies for the precise specification of the problem to solve with the machine, establish methods for the detailed and accurate description of solutions that can be implemented on a computer. For this, the student will also have to learn one or more programming languages and their paradigms, in order to use the notions of programming and systematize the use of data structures and algorithms to solve different categories of problems.

Failure, demotivation and dropout rates are traditionally high in introductory curriculum units. [2] [3] [4]. Some even hold that students cannot program at the end of introductory units [5]. According to Winslow [6] Psychological studies of specialization in general and computer programming knowledge in particular show that turning a novice into an expert is impossible in a four-year program. Competence, however, is possible. Achieving this level requires mastery of facts, characteristics, and ground rules, and the ability to plan and troubleshoot in specified areas. The key to reaching this level is practice, practice, practice - starting with simple facts and problems and working towards increasingly complicated facts, strategies and problems.

Often our expectations appear to be wrong: “Our expectations may have been too high, the problems may have been too difficult or too little for students' education and interests, there may not be enough

time and so on." [5]; "Typically, students are expected to be able to write small programs that minimally use conditions, cycles, and arrays. They are expected to solve problems using functional decomposition and write programs involving various methods. Although they are often not explicitly evaluated, we also often expect students to gain familiarity, if not complete experience, with the systems and tools used during programming. We expect a lot, and the evidence indicates that many, if not most, students cannot meet our expectations." [7]. And what is expected at the end of the course? McCracker et al. [5] describe the students' competences: "1. Abstract the problem of their description; 2. Generate sub problems; 3. Turn sub-problems into sub-solutions. 4. Compose the sub-solutions in a program again; 5. Evaluate and repeat.

Many argue that all introductory programming courses teach the same thing: computer programming basics. From this perspective, there would be no need for papers describing the curriculum of an introductory programming course. Although most work on introductory programming seems to accept the curriculum as given, there are others for which aspects of the curriculum are the focus. [8].

Three studies show how much research and production there is in this area.

Valentine [9] did a twenty-year meta-analysis of SIGCSE (Special Interest Group for Computer Science) for CS1 and CS2 and describes a six-point article taxonomy to classify 444 presentations from 1984-2003: Marco Polo, Tools, Experimental, Stylish, Philosophy, and John Henry. The description of the use of languages in CS1 will start from Marco Polo ("I went there and I saw this.") and represents about 30% of the publications of the first decade and 24% of the second.

Becker and Quille [10] have done a trend analysis of introduction to programming over the past 50 years, using articles presented at the SIGCSE conferences that focus on early programming and CS1. They analyzed 481 articles related to CS1 and organized them into 8 categories: languages and paradigms; CS1 design, structure and approach; CS1 content; tools; collaborative approaches; teaching; Learning and assessment; students. These eight categories were further divided into 54 subcategories (for example the first category was divided into 4: general languages and paradigms; specific paradigms; specific text-based languages; others).

The document "Introductory Programming: A Systematic Literature Review" [8] cites 700 references from 2003 to 2017 found from "introductory programming" OR "introduction to programming" OR "novice programming" OR "novice programmers" OR "CS1" OR "CS 1" OR "learn programming" OR "learning to program "OR" teach programming ". Initially they used a rating for 1666 articles with 4 student categories, teaching, curriculum and assessment. These 4 categories were further subdivided into several others (curriculum was divided into skills, programming languages, paradigms). We found 70 articles on the topic programming language choice, ie 10% of the total articles analyzed.

2 EVOLUTION OF THE CHOICE OF INITIAL PROGRAMMING LANGUAGES, PAST CENTURY

Computer science became a recognized academic field in October 1962 with the creation of Purdue's first department [11]. 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 [12] with early indications of curriculum models for programs in computer science and computer engineering. This report defined a "classification of subject areas contained in computer science and described twenty-two courses in these areas." Prerequisites, catalog descriptions, detailed sketches, and annotated bibliographies were included for each of these courses. It had a module called B1-Introduction to computing (2-2-3) in which an algorithmic language was proposed, recommending that only one language be used, or more "in order to demonstrate the great diversity of available computer languages. Due to its elegance and novelty, SNOBOL can be used quite effectively for this purpose".

With the emergence of a host of new courses and departments, ACM published a new report, Curriculum '78: recommendations for the undergraduate program in computer science [13], with Curriculum 68 updates. In this document "the basic curriculum common to all computer science degree programs is presented in terms of topics and elementary and intermediate level courses". It first introduced CS1: Computer Programming I (2-2-3): the initial course with an emphasis on the techniques of developing and programming stylish algorithms. "Neither the esoteric features of a programming language nor other aspects of computers should interfere with that goal."

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 [14] in order to “detail a first course in computer science that emphasizes programming methodology and problem solving.” In this report we refer to Pascal, PL / 1 and Ada: “These features are important for several 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. ” It is noted that FORTRAN and BASIC, "although widely used are not suitable for CS1" and that ALGOL "meets the requirements but is no longer widely used or supported." Computing as a discipline [15] is an article that aims to give more importance to other disciplines than programming: “computer science encompasses far more than programming”.

In 1991 [16] IEEE (Institute of Electrical and Electronics Engineers) and ACM have joined for a new document. This emerged by breaking with some of the concepts from previous documents, presenting a set of individual knowledge units corresponding to a topic that should be addressed at some point to the undergraduate. In this way, institutions have considerable flexibility in setting up course structures that meet their particular needs.

In the early days FORTRAN was selected as a high level language for 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. [17]. At that time you couldn't talk about methodology: everything was just programming. The emergence of BASIC in 1964 led some departments to use this language for introductory students [18]. In 1972 [19] 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 1960s, some departments experimented various languages such as PL / I.

With Dijkstra's manifesto [20] structured programming begins to be discussed [21] [22]. The emergence of Pascal [23] seems to become almost consensual [17]: a language written with the goal of learning programming with a very friendly development environment as well as support and documentation [24], and obviously the proliferation of personal computers and the availability of Pascal compilers [25].

3 THE EVOLUTION OF PROGRAMMING LANGUAGES FROM THE 90S

Pascal's decline began in the late 1980s, early 1990s, with object-oriented programming. And also because the language has difficult document reuse but also because Pascal is not a "real world" language [25]. McCauley and Manaris [26] report 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. In this study, Residual Visual Basic, Scheme and Modula-2 appear.

In 2001 a new document was published [27] where programming-first from previous documents is questioned 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 “Focus on programming, excluding of other topics, it gives students a sense of discipline, thus reinforcing the common misperception that “computer science” equals programming”. It is suggested that the sequence CS1 and CS2 move to three units: 101, 102 and 103 or in the case where 111 and 112 is not feasible. It is said that “The programming approach can first be exacerbated in the first object model. First, because many of the languages used for object-oriented programming in the industry - particularly C ++, but to some extent Java as well - are significantly more complex than classic languages. Unless teachers take special care to introduce material in a way that limits this complexity, these details can easily overwhelm introductory students. ”

In 2008 a new document was published: “Computer Science Curriculum 2008: An Interim Revision of CS 2001” [28] where security is heavily addressed, making minor revisions to the 2001 paper. It reinforces the idea that “Computer science professionals often use different programming languages for different purposes and should be able to learn new languages during their careers. As a result, students must recognize the benefits of learning and applying new programming languages. It is also important for students to recognize that choosing the programming paradigm can significantly influence how they think about problems and express solutions to those problems. To this end, we believe that all students must learn to program in more than one paradigm. ”

Referring to languages and paradigms, “Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science” [29] says that the choice of programming languages is dependent on the paradigm chosen and that “there seems, however, to be a growing trend

towards “safer” or more managed languages as well as the use more dynamic languages such as Python or JavaScript.” As for Visual programming languages, such as Alice and Scratch, it is recommended that they be used only with non-graduates.

Farooq et al. [30] present by percentage and years the initial programming languages used where it is clearly shown that Pascal was used by 40% in 1994 but by 0% in 2006, and that in 2011 it was noted that Java and C ++ were the most used (79% together).) but that Python was appearing.

In Portugal [31], 2016-2017, the most common first-year programming language sequence in 46 courses analyzed was just C (48%), followed by just java (22%), C + Haskell (9%), C + java (4%), Scheme + java (4%). There were also residual sequences of Excel + C, just Python, Python + HTML + java, Python + java, Scheme + C ++ and XML + java. Regarding the 10 most significant Portuguese Computer Engineering courses [32], the most common sequences were Java + java and Python + C (30%), C + C (20%), Python + java (10%) and Haskell + C (10%).

According to the document “An Analysis of Introductory Programming Courses at UK Universities” [33]: the most commonly used programming language is Java (46%), followed by the “family” C (C, C ++ and C #) (23.6%) and Python (13.2%). Javascript and Haskell are much less used.

According to the document “Introductory Programming Courses in Australasia in 2016” [34], Concerning Australian and New Zealand Universities: Of the 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.

In “What language? - The choice of an introductory programming language” [35] a study of 496 four-year courses in the United States was done and it was found that Java was used by 41.94%, Python 26.45%, C ++ 19.35%, C 4.52%, C # 0.65%, and another 7.10%.

A 2016 study [36] analyzing 218 colleges and 143 universities in 35 European countries indicates that the most commonly used programming language was C (30.6%), followed by C ++ (21.9%) and Java (20.7%).

One document [37] with 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%) ”.

4 A CONTRIBUTION WITH GOOGLE SCHOLAR

4.1 Method

We reviewed the Google Scholar site [38] which has a huge amount of articles.

We use several types of parameters for the search:

a) cs1 OR "introductory programming" OR "introduction to programming" OR novice AND "programming language".

b) cs1 OR "introductory programming" OR "introduction to programming" OR novice AND "programming language" and each of the programming languages to be studied and excluding the others.

4.2 All Programming Languages

The search for only cs1 OR "introductory programming" OR "introduction to programming" OR novice AND “programming language" returns 39221 papers. There is a growing number of papers in Google Scholar each year.

Table 1. cs1 OR "introductory programming" OR "introduction to programming" OR novice AND “programming language"

Year	Papers
1989	463
1990	570
1991	494
1992	603
1993	563

1994	618
1995	586
1996	647
1997	617
1998	677
1999	660
2000	747
2001	809
2002	827
2003	1100
2004	1350
2005	1270
2006	1640
2007	1610
2008	1610
2009	1600
2010	1800
2011	1860
2012	2070
2013	2170
2014	2390
2015	2440
2016	2440
2017	2610
2018	2380
All	39221

4.3 By programming languages

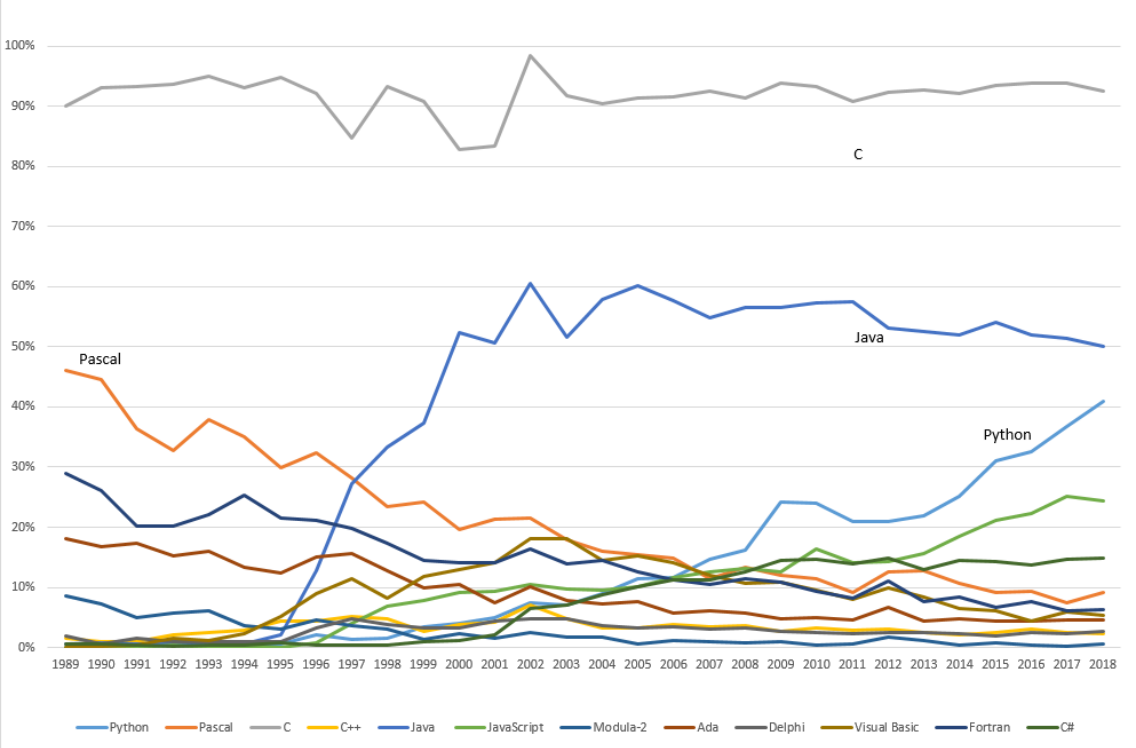
The following table and graphs list the percentage per year of documents in which each of the languages appear. We can see the importance of the C language and java language that appear in more than 39000 Google Scholar platform documents with cs1 OR "introductory programming" OR "introduction to programming" OR novice AND "programming language". Pascal declines against Python evolution.

Table 2. Google scholar by programming language

	Python	Pascal	C	C++	Java	JavaScript	Modula2	Ada	Delphi	Visual Basic	Fortran	C#
1989	0%	46%	90%	2%	1%	0%	9%	18%	2%	0%	29%	1%
1990	0%	45%	93%	1%	1%	0%	7%	17%	1%	0%	26%	1%
1991	1%	36%	93%	1%	1%	0%	5%	17%	2%	0%	20%	0%
1992	0%	33%	94%	2%	1%	0%	6%	15%	1%	1%	20%	0%
1993	0%	38%	95%	2%	0%	0%	6%	16%	1%	1%	22%	0%
1994	0%	35%	93%	3%	1%	0%	4%	13%	1%	2%	25%	0%
1995	1%	30%	95%	4%	2%	0%	3%	12%	1%	5%	22%	1%
1996	2%	32%	92%	4%	13%	1%	5%	15%	3%	9%	21%	0%
1997	1%	28%	85%	5%	27%	4%	4%	16%	5%	12%	20%	0%
1998	2%	23%	93%	5%	33%	7%	3%	13%	4%	8%	17%	0%

1999	3%	24%	91%	3%	37%	8%	1%	10%	3%	12%	15%	1%
2000	4%	20%	83%	4%	52%	9%	2%	10%	3%	13%	14%	1%
2001	5%	21%	83%	4%	51%	9%	2%	7%	4%	14%	14%	2%
2002	7%	22%	98%	7%	60%	10%	3%	10%	5%	18%	16%	6%
2003	7%	18%	92%	5%	52%	10%	2%	8%	5%	18%	14%	7%
2004	9%	16%	90%	3%	58%	9%	2%	7%	4%	15%	14%	9%
2005	11%	15%	91%	3%	60%	10%	1%	8%	3%	15%	13%	10%
2006	12%	15%	91%	4%	58%	12%	1%	6%	4%	14%	11%	11%
2007	15%	11%	93%	3%	55%	13%	1%	6%	3%	12%	11%	11%
2008	16%	13%	91%	4%	56%	13%	1%	6%	3%	11%	11%	13%
2009	24%	12%	94%	3%	57%	13%	1%	5%	3%	11%	11%	15%
2010	24%	12%	93%	3%	57%	16%	0%	5%	2%	10%	9%	15%
2011	21%	9%	91%	3%	58%	14%	1%	5%	2%	8%	8%	14%
2012	21%	13%	92%	3%	53%	14%	2%	7%	2%	10%	11%	15%
2013	22%	13%	93%	3%	53%	16%	1%	4%	3%	8%	8%	13%
2014	25%	11%	92%	2%	52%	19%	0%	5%	2%	6%	8%	15%
2015	31%	9%	93%	3%	54%	21%	1%	4%	2%	6%	7%	14%
2016	33%	9%	94%	3%	52%	22%	0%	4%	3%	4%	8%	14%
2017	37%	7%	94%	2%	51%	25%	0%	5%	2%	6%	6%	15%
2018	41%	9%	92%	2%	50%	24%	1%	5%	3%	5%	6%	15%

Figure 1. *Google scholar by programming language.*



5 CONCLUSIONS

This article makes a historical review of the programming languages used in the introduction to computing over the last 30 years, differentiating an evolution of programming language choices in the past century and now. At the end of the article, an evolutionary trend is listed by articles published by people involved in the subject.

The methodology used in this article compiles the Google Scholar articles for each of the last 30 years (1988-2018) and analyses which programming languages are used in academic studies. It is very interesting to see how the programming languages used in the introductory programming units change over the years: the Pascal decline and the appearance of Python, but also the maintenance of C and Java as programming languages used by the CS1 community.

REFERENCES

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

[2] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," vol. 39, no. 2, pp. 32-36, 2007.

[3] A. Yadin, "Reducing the dropout rate in an introductory programming course," *ACM Inroads,* vol. 2, no. 4, pp. 71-76, 2011.

[4] B. A. Becker, C. Murray, T. Tao, C. Song, R. McCartney and K. Sanders, "Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages," in *49th ACM Technical Symposium on Computer Science Education*, 2018.

[5] 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.

[6] L. E. Winslow, "Programming pedagogy—a psychological overview," *ACM SIGCSE Bulletin,* vol. 28, no. 3, pp. 17-22, 1996.

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

[8] A. S. ,. A. I. B. B. G. M. K. A. O. L. P. J. S. M. S. J. a. S. C. Luxton-Reilly, "Introductory Programming: A Systematic Literature Review.," in *roceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018.

[9] D. W. Valentine, "CS educational research: a meta-analysis of SIGCSE technical symposium proceedings," in *SIGCSE Technical Symposium*, 2004.

[10] B. A. Becker and K. Quille, "50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research," in *50th ACM Technical Symposium on Computer Science Education*, 2019.

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

[12] 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.

[13] 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.

[14] 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.

[15] P. J. Denning, "Computing as a discipline," *Communications of the ACM,* vol. 32, no. 1, pp. 9-23, 1989.

[16] 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 , 1991, p. 154.

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

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

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

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

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

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

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

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

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

[26] 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.

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

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] R. Mason and Simon, "Introductory Programming Courses in Australasia in 2016," in *Nineteenth Australasian Computing Education Conference*, 2017.

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

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

[37] 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.

[38] Google, "Google Scholar," Google, set 2019. [Online]. Available: https://scholar.google.com/.

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

[40] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin and C. Mooney, "Effective Compiler Error Message Enhancement for Novice Programming Students," *Computer Science Education,* pp. 148-175, 2016.

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

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

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

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

[45] 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.

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

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

[48] 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.

[49] 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.

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