Programmers' Affinity to Languages

Alvaro Costa Neto
Federal Institute of Education, Science and Technology of São Paulo, Barretos, Brazil

Cristiana Araújo
Centro ALGORITMI, Departamento de Informática,
University of Minho, Campus Gualtar, Braga, Portugal

Maria João Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics,
Polytechnic Institute of Bragança, Portugal

Pedro Rangel Henriques
Centro ALGORITMI, Departamento de Informática,
University of Minho, Campus Gualtar, Braga, Portugal

Abstract

Students face several challenges when learning computer programming languages, a central topic to acquire programming skills. While those challenges that present a predominantly technical nature have been intensely studied by researchers along the years, the ones that are concerned with qualitative, and personal aspects have not. Affinity to a programming language is one of the many personal factors that may contribute to surpass these qualitative aspects that describe the difficulties that students face. From this point-of-view, this paper presents a proposal for treating and studying programmers' affinity to programming languages as an important factor for learning computer programming. It also reports a preliminary questionnaire conducted on a master's degree class at Universidade do Minho that showed that affinity may have a broader relation to learning computer programming than anticipated. Finally, a set of relevant questions are stated to compose a future inquiry aimed at deepening the knowledge on the affinity between programmers and languages, paving the way for following research.

2012 ACM Subject Classification Applied computing → Computer-assisted instruction

Keywords and phrases Computer programming, Programming Languages, Affinity, Education, Learning

Digital Object Identifier 10.4230/OASIcs.ICPEC.2021.3

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Learning a complex topic such as computer programming is an intricate process, influenced by several factors of varied nature. Research in this field dates back several decades [5], and has kept constant academic interest throughout the years [6, 15, 12, 3], usually through a technical approach – development and evaluation of tools that are used to teach and learn. While of great value, these are not the only problems that students face when learning. In fact, when observed through a pedagogical lenses, many other characteristics of different nature also seem to explain how knowledge is constructed, allowing food for thought on how to facilitate this process in a more personal perspective. The widely known researcher Jean Piaget stated that context, especially in its social branch, has a decisive impact on learning by providing the necessary interactions that adapt and organize the mental schemes [11]. Lev Vygotsky, another very influential pedagogy researcher, also highlighted the crucial importance that contextual factors play in the construction of knowledge [14], while positioning language as the main mediator for learning. Paulo Freire went one step further by supporting the idea that teaching is a force of change to socioeconomic contexts, which is only possible by understanding and aggregating the students' backgrounds into the process [7]. In a different recent project [4], focused in adults' professional reconversion, we also realized the relevance of background and know-how in their learning process, including the impact on the choice of the best learning resources. Based on these authors' findings, it's reasonably safe to assume that personal factors are inevitably involved in the process of knowledge construction, and therefore should be taken into account when teaching.

As one of these personal factors, *affinity* may sustain an important relation to the learning process, even on very technical subjects such as computer programming, and it deserves, for sure, to be further analysed. While there have been studies on personal factors being influential to the education process as a whole, affinity's influence is still largely unexplored when it comes to computer programming learning. Affinity's qualitative nature and the fact that it may be directly related to technical aspects of the programming languages present a dual, albeit interesting conundrum to investigate. As such, many research questions emerge when affinity is considered an important feature of the learning process:

- Which of the language's characteristics stimulate students' subconsciousness, subtly conducting them to like or dislike certain programming languages?
- Do these characteristics influence students positively or negatively?
- Is the inverse actually true, meaning that the first programming language learnt strongly influences the consequent student's affinity with that particular language?

Finding answers to these questions is the main goal of the study discussed in this paper, aiming to instigate further research on this subject.

This paper is structured in five sections: after the introduction, section 2 discusses which factors may influence the learning process and how they may be categorized. The third section presents affinity as an important and challenging topic for discussion when considering the learning of computer programming. Moreover, in this section it is also presented the initial investigation that underlies the work here reported. Based on the previously presented results, section 4 raises relevant questions that must be answered in order to better understand the role of affinity in learning, and in the development of programming skills. Finally, section 5 concludes this paper presenting a proposal of further investigation that will be conducted.

2 Influences on Learning Computer Programming

Many factors influence directly or indirectly the learning process. Whenever possible, it is good practice to take into account these influences in order to improve the student's experience. Three main categories may be established: teaching methods, programming languages characteristics and personal background.

The *teaching methods* category contains the array of factors based on the application of pedagogical methods, either formally stated or experience based. Tools and other mediating resources are also included in this category, such as proposed by [1, 2]. The most common factors include:

- Quality of the communication process;
- Media diversification;
- Auxiliary tools;
- Clarity of the explanation *etc.*

Factors that are directly connected to the underlying principal and design of the programming language are its formal definition, paradigm, syntax, semantics, and quality [10]. Also important are intrinsic characteristics such as:

- Expressiveness;
- Viscosity;
- Verbosity
- Readability;
- Consistence;
- Error-proneness;
- Abstraction level *etc.*

Historical, contextual, and subconscious characteristics form the *personal background* category. Examples of such factors include:

- Socioeconomic context;
- Previous contact with formal logic and computers;
- Experience with the English language [9];
- Psychological issues such as being away from family for the first time [8];
- Affinity with specific languages and tools;
- Lack of interpersonal skills *etc.*

Figuring in as a personal and complex factor, *affinity* may be established by the students to specific programming languages. On one hand, considering affinity may lead to new paths to motivate students improving their learning process. On the other hand, affinity might also be seen as a result of the students' experiences, guided by their courses' curricula and the initial languages he or she has learnt.

3 Affinity to Programming Languages

The previous statements are supported by a preliminary indication observed by asking twenty three students of a Masters' degree class in Computer Engineering at Universidade do Minho to answer a few questions about programming languages and their learning experiences. After a lecture on teaching computer programming, in which many of these topics were discussed, some snippets of code were shown in different languages – BASIC, Lisp, C, Java, Python, Ruby, Swift, and Haskell – and students were asked to point out which languages they like most, being C, Java and Haskell the three highest ranked. Then, a questionnaire was applied containing these questions:

1. In a range of very low to very high, how important is the language choice for learning computer programming? Justify your answer;
2. Out of those factors previously mentioned in Section 2 and presented in the lecture, which ones are most relevant, and influential to learning computer programming?
3. Which languages would you choose to teach computer programming: BASIC, Lisp, C, Java, Python, Ruby, Swift, or some other language? Justify your answer. This was a multiple choice question, allowing students to choose more than one language.

If popularity and market share were the main influences driving the students' choices, it would be safe to assume that Python would be the highest ranked option, since it's currently a very popular programming language [13], with many popular applications, such as artificial intelligence and numerical computing. Nonetheless, C and Java overthrew Python as shown in Figure 1. Haskell also had a perceptive presence in the results, which is also unexpected from the perspectives of current market share and ease of use. These surprises become understandable if the affinity to the language was actually constructed as a *consequence of their learning experiences*, since these students had been formally taught Haskell, C and Java as their initial programming languages. While the current attractiveness of Python can't be discarded – it ranked third, after all – these results show that programming language affinity is more subtle and complex than expected, and could ultimately influence other activities beyond learning.

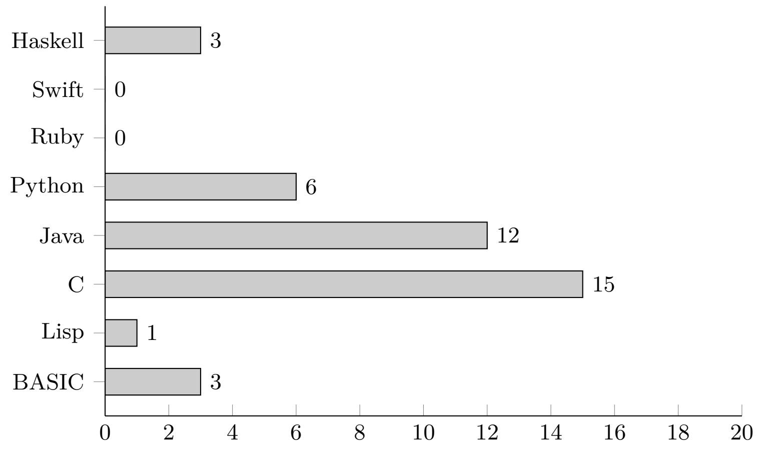

Figure 1 Language choices by students when asked which languages they prefer.

These unexpected findings imply that many questions should arise from more exploratory observation of how students learn computer programming, creating opportunities for deeper investigation on affinity with programming languages and its relation to the learning process.

4 A Survey to Understand Affinity

Learning how to correctly program a computer involves not only understanding how to type commands that are syntactically correct. It also involves understanding problems, abstracting concepts and finding recurrent patterns, delineating their boundaries, designing algorithms, and translating viable solutions to source code. In this process, the programming language acts simultaneously as a tool and as an interface of communication, interfering with both learning and problem solving. Having a close understanding of its rules is mandatory for obtaining good results, but solving a problem using a language enacts a more complex intellectual mechanism, specially when learning.

It becomes clear that the mental schema that must be constructed in order to program an application involves many aspects of the human mind: memory, perceptiveness, and motivation, to name a few. As is safe to assume that affinity is an important part of motivation, its proper usage can be a valuable aid for education professionals. Several interesting questions arise to help understanding affinity and the role it might play in learning computer programming:

- How does affinity to a programming language come to be?
- Which factors affect its growth, either hampering or stimulating it?
- Is this affinity a support to or a consequence of the learning process?
- How do teaching methodologies come into play to stimulate it?
- Which personal factors affect affinity growth?

These questions, while of central importance, are too broad to directly compose a survey. They serve to structure the objectives by instigating the search for their answers. In order to obtain tangible results though, it's necessary to delve deeper into the problem of affinity, and pinpoint specific characteristics through a more palpable inquiring. This process, in and of itself, proposes an interesting challenge: how to obtain meaningful answers for a learning factor that may present itself simultaneously technical and personal, such as affinity?

This duality of nature may require an equivalent dual approach. While it's important to directly ask questions such as "Is this language appealing to you? Why?", or "Rank your favorite languages" to paint a personal picture of the respondent's background, this strategy may not work if applied to more technical aspects of the programming languages. In this case, direct questions may lead to colored answers, either by previous personal opinions, or by misinterpretation of what is being asked. A dual approach would apply indirect observation of affinity on technical aspects – comparing snippets of source code, would be an example – while still relating these aspects to the respondent's background, via more open, direct, and personal questions. This approach seems to be a more encompassing way to obtain meaningful answers, at least for the previously stated questions.

According to the previous discussion, we propose a survey based on the following structure:

1. A collection of personal questions in order to capture a picture of the respondents' contexts and backgrounds. It would also be interesting to ask directly which languages are their favorites and why, for later comparison with the answers given to the other questions;
2. Direct questions asking the respondents to analyze snippets of source code in various programming languages. These questions should be as direct and as clear as possible to avoid misinterpretations, and to accommodate the fact that not all respondents may be familiar with all presented languages. Several different languages should be applied – randomly, if possible – in order to assess their capacities of understanding source code, while simultaneously identifying direct and indirect characteristics that emerge as possible influential factors for the respondents' affinity to the languages that were chosen as their favorites;
3. A series of indirect questions, also applying several different programming languages and snippets of source code. These questions must be composed in such a way to detect indirectly which characteristics of the presented code are influencing the respondents' affinity to the presented languages, given their backgrounds and current knowledge;
4. Some direct questions inquiring what is his or her perception of the languages and snippets previously presented. These questions would serve as a leveling reference for the respondents' personal – and biased – view of the programming languages and what he or she thinks is more influential to create affinity. The answers for these questions are certain to form interesting comparisons with the ones given to the questions proposed on item 1.

This structure will support the construction of the survey. Implementation details, such as the order of the questions, which programming languages should be used, and complexity of the source code snippets, are yet to be determined. It would be ideal to apply dynamically constructed surveys, that would serve two main purposes:

- To provide specific inquiries for respondents that have different backgrounds, in order to minimize personal bias on more direct and technical questions, and;
- To design some mechanism of comparison between respondents that have experience with certain languages and those who have not.

The technical resources to implement these dynamic surveys are yet to be investigated and developed, if required. While having education purposes in mind, the survey will be freely available, and account for answers given by people other than students and teachers. Computer programming professionals, mathematicians, applied computing users, to list a few, may have very interesting, and important, views on affinity to their favorite programming languages.

Finally, the answers will be statistically correlated to the respondents' background and technical knowledge in order to obtain a better understanding of how and why affinity to a programming language is established. The final goal is to identify which characteristics of the current, and past programming languages are most related to the rise and fall of programmers' affinity. This information may be a valuable resource to aid educators to better choose – or when possible, allow the choice – of the programming language used in their classes.

5 Conclusion

The problem of more efficiently teaching computer programming still poses relevant, and appealing challenges. While many studies have shown diverse characteristics as influential to the learning experience, affinity to a programming language is still a largely unknown aspect.

Preliminary results point toward further investigation, as the conducted studies show that there are at least two – probably complementary – perspectives on affinity: it may be viewed both as a propelling factor to knowledge construction, and as a result of experience while learning computer programming. The former indicates that educators must be aware of the relationship that students develop with the programming languages they taught, specially in their initial stages of education. This observation, and consequent adaptation of the teaching process should improve students' learning experiences. The later – considering affinity as a product of the learning experience itself – may become a valuable indication to better understand how the programming languages taught in formal education influence the industry. On one hand, this affinity may induce future professionals – and consequently, their workplaces – to prefer and use the languages they were taught. On the other hand, it may create a resistance in the workforce to adopt current established languages. Nonetheless, in both perspectives affinity should be considered an important factor of the teaching-learning process, and consequently be taken into account when deciding which languages students should learn, work and improve.

In the near future deeper studies shall be conducted, investigating which specific characteristics contribute to language affinity, both technical, and personal in nature.

References

1 M. V. P. Almeida, L. M. Alves, M. J. V. Pereira, and G. A. R. Barbosa. EasyCoding - Methodology to Support Programming Learning. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.1.

2 M.V.P. Almeida. Easycoding: Methodology to support programming learning. Master's thesis, Instituto Politécnico de Bragança, 2020.

3 A.G. Applin. Second language acquisition and cs1. *SIGCSE Bull.*, 33(1):174–178, February 2001. doi:10.1145/366413.364579.

4 D.R. Barbosa. CnE-Ar: Teaching of Computational Thinking to Adults in Reconversion. Master's thesis, Minho University, Braga, Portugal, April 2021. MSc dissertation (to be discussed and published).

5 R.R. Fenichel, J. Weizenbaum, and J.C. Yochelson. A program to teach programming. *Communications of the ACM*, 13(3):141–146, March 1970. `doi:10.1145/362052.362053`.

6 J. Figueiredo and F.J. García-Peñalvo. Building skills in introductory programming. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'18, page 46–50, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3284179.3284190`.

7 P. Freire. *Pedagogia da Autonomia: Saberes necessários à prática educativa.* Paz e Terra, 2011.

8 A. Gomes and A.J. Mendes. Learning to program: difficulties and solutions. In *Proceedings of the 2007 ICEE International Conference on Engineering and Education*, ICEE '07. International Network on Engineering Education and Research, 2007.

9 P.J. Guo. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–14, New York, NY, USA, 2018. Association for Computing Machinery.

10 P.R. Henriques. *Brincando às Linguagens com Rigor: Engenharia gramatical.* PhD thesis, Universidade do Minho, 2013.

11 J. Piaget, M. Piercy, and D.E. Berlyne. *The Psychology of Intelligence.* Routledge classics. Routledge, 2001.

12 S.A. Robertson and M.P. Lee. The application of second natural language acquisition pedagogy to the teaching of programming languages—a research agenda. *SIGCSE Bulletin*, 27(4):9–12, December 1995. `doi:10.1145/216511.216517`.

13 StatisticsTimes.com. Top computer languages, 2020. URL: `http://statisticstimes.com/tech/top-computer-languages.php`.

14 L.S. Vygotsky, E. Hanfmann, G. Vakar, and A. Kozulin. *Thought and Language.* The MIT Press. MIT Press, 2012.

15 B.C. Wilson and S. Shrock. Contributing to success in an introductory computer science course: A study of twelve factors. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '01, page 184–188, New York, NY, USA, 2001. Association for Computing Machinery. `doi:10.1145/364447.364581`.