Towards Supporting Knowledge Transfer of Programming Languages

Nischal Shrestha
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
nshrest@ncsu.edu

I. Introduction

Today, there are hundreds of programming languages that are widely used. Programmers at all levels are expected to become proficient in multiple languages. Experienced programmers who have knowledge of at least one language are able to learn a second language much quicker than novices. However, the transfer process can still be difficult when there exists numerous differences from their previous language. Documentation, online courses and tutorials tend to present information geared towards novices. This type of presentation might suffice for beginners, but it doesn't support learning for experienced programmers [1] who would benefit from leveraging their knowledge of previous programming languages.

In my work, I explore teaching programming languages through the lens of *learning transfer*, which occurs when learning in one context either enhances (positive transfer) or undermines (negative transfer) a related performance in another context [2]. To investigate this approach, I created and evaluated a research tool called Transfer Tutor that teaches programmers R in terms of Python and Pandas, a data analysis library (see Fig. 1). The following design choices were made to explore learning transfer, applied to the topic of data frame manipulation: 1) highlighting similarities between syntax elements to support learning transfer 2) explicit tutoring on potential misconceptions 3) stepping through and highlighting elements of the snippets incrementally.

There are few studies examining transfer in the context of programming languages. Transfer of declarative knowledge between programming languages has been studied by Harvey and Anderson [3], which showed strong effects of transfer between Lisp and Prolog. Scholtz and Wiedenbeck [4] found that programmers suffer from negative transfer of Pascal or C knowledge when implementing code in a new programming language called Icon. Wu and Anderson [5] found problem-solving transfer for programmers writing solutions in Lisp, Pascal and Prolog which could improve programmer productivity. However, none of these studies investigated tool support.

Bower [6] explored a new teaching approach called Continual And Explicit Comparison (CAEC) to teach Java to students who knew C++. They found that students benefited from the continual comparison of C++ concepts to Java. Transfer Tutor uses a similar teaching approach but provides interactivity, allowing programmers to visualize the differences between two languages with the use of highlights, which serves as affordances for transfer [7]. Further, Transfer Tutor allows programmers to step through the code, which helps them make a mindful abstraction of the concepts [8].

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

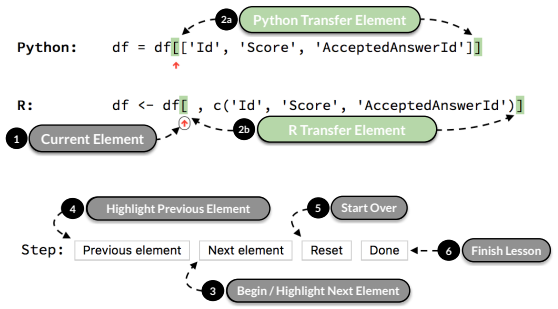

Fig. 1: The red arrow (1) indicates the currently highlighted syntax element for the code snippets in Python (2a) and R (2b). The stepper buttons are used to start the lesson or step forwards (3) and backwards (4) through the relevant syntax elements of the code snippets, reset (5) or end the lesson (6).

Fix and Wiedenbeck [9] developed and evaluated a tool called ADAPT that teaches Ada to programmers who know Pascal and C. Their tool helps programmers avoid implementation plans which contain negative transfers from Pascal and C, but is targeted primarily at the planning level. Transfer Tutor explicitly emphasizes both syntax and semantic issues by highlighting differences between the syntax elements in the code snippets of the two languages. Unlike ADAPT, Transfer Tutor is focused on transferring declarative knowledge [3], such as syntax rules, rather than procedural knowledge, such as implementation planning.

II. Preliminary findings

I conducted a user study of Transfer Tutor with 20 participants from a graduate Computer Science course at North Carolina State University. A qualitative analysis on think-aloud protocols revealed that participants made use of learning transfer even without explicit guidance. The responses to a user satisfaction survey revealed additional insights on the design implications of future tools supporting transfer:

A. Affordances for supporting learning transfer

The majority of the participants found that incrementally stepping through the syntax elements was a useful feature as it helped them focus on one syntax element at a time and catch misconceptions on the spot. However, it prevented more advanced programmers from easily skipping explanations from the tool. Despite the usefulness of always-on visualizations in programming environments [10], allowing the programmer to activate explanations on-demand might be beneficial, using a mouse hover for example.

Reducing information load and allowing live code execution were two improvements suggested by the participants. This suggests Transfer Tutor needs to reduce information overload and balance the volume of explanation against the amount of code to be explained. One solution is to externalize additional explanations to documentation outside of the tool, such as web resources. Breaking up lessons into smaller segments could also reduce the amount of reading required. Future iterations of Transfer Tutor could include code execution, adapting explanations for the programmer's code.

B. Expert learning can benefit from learning transfer

The *expertise reversal effect* suggests that instructional techniques that are effective for novices can have negative consequences for experience learners [11]. I have tried mitigating this effect by presenting explanations in terms of language transfer—in the context of a language that the programmer is already an expert in. Transfer Tutor serves as an instructional intervention: experienced programmers can use the tool to familiarize themselves with the language and over time reduce and eventually eliminate use of the tool.

III. Future Work

A. Automatic code translation and annotation

Transfer Tutor lacks support for easily finding the mappings between the syntax of two programming languages. The lesson designer needs to manually translate code from one language to another which is a tedious and error-prone process. Automatic code translation of code would help ease this process. For example, SMOP (Small Matlab and Octave to Python compiler)[1] is a transpiler that converts Matlab/Octave code to Python which is useful for code reuse. However, the resulting Python code is not useful for learning purposes as the programmer still needs to relate Python back to Matlab or Octave. The tool would be more useful if the generated code also contained annotations of the translation that took place so that programmers can better understand Python in relation to Matlab/Octave.

B. Mining for transfer issues

Researchers and instructors may find it difficult to identify the most important transfer issues. This could be solved by mining Q&A sites like Stack Overflow (SO). Using Truede's methodology [12], I performed a preliminary analysis on SO posts tagged with both ⟨R⟩ and ⟨Pandas⟩. I discovered that most programmers ask about how to translate a piece of code in Python/Pandas to R or vice-versa. Most accepted answers to these questions provide not only the equivalent piece of code in the target language, but also rich explanations that describe exceptions and gotchas. Mining could be a useful approach for collecting empirical data on transfer issues across programming languages.

[1] https://github.com/victorlei/smop

C. Automatic design of transfer lessons

Currently, there is no easy method to create new transfer lessons for programming languages. Future tools could allow an instructor to specify the source and target language to generate a series of lessons automatically. If teaching Python, for example, the tool would first cover fundamental topics like variable assignment or for loops then slowly scale up to more advanced topics like list comprehensions. It would also be helpful if the tool generated interactive code examples and quizzes for each lesson to help learners test their knowledge. This addresses the limitations of Transfer Tutor regarding the lack of hands-on experience and the manual labor required to design a series of lessons. The two previous approaches–mining and automatic translation–serve as supplemental methods for this research direction.

IV. Acknowledgements

I would like to thank my advisor, Dr. Chris Parnin, for his advice and support of this work. This material is based in part upon work supported by the National Science Foundation under Grant Nos. 1559593 and 1755762.

References

[1] L. M. Berlin, "Beyond program understanding: A look at programming expertise in industry," *Empirical Studies of Programmers (ESP)*, vol. 93, no. 744, pp. 6–25, 1993.

[2] D. N. Perkins, G. Salomon, and P. Press, "Transfer of learning," in *International Encyclopedia of Education*. Pergamon Press, 1992.

[3] L. Harvey and J. Anderson, "Transfer of declarative knowledge in complex information-processing domains," *Human-Computer Interaction*, vol. 11, no. 1, pp. 69–96, 1996.

[4] J. Scholtz and S. Wiedenbeck, "Learning second and subsequent programming languages: A problem of transfer," *International Journal of Human–Computer Interaction*, vol. 2, no. 1, pp. 51–72, 1990.

[5] Q. Wu and J. R. Anderson, "Problem-solving transfer among programming languages," Carnegie Mellon University, Tech. Rep., 1990.

[6] M. Bower and A. McIver, "Continual and explicit comparison to promote proactive facilitation during second computer language learning," in *Innovation and Technology in Computer Science Education (ITiCSE)*, 2011, pp. 218–222.

[7] J. G. Greeno, J. L. Moore, and D. R. Smith, "Transfer of situated learning." in *Transfer on trial: Intelligence, cognition, and instruction.* Westport, CT, US: Ablex Publishing, 1993, pp. 99–167.

[8] D. H. Schunk, "Learning theories," *Printice Hall Inc., New Jersey*, pp. 1–576, 1996.

[9] V. Fix and S. Wiedenbeck, "An intelligent tool to aid students in learning second and subsequent programming languages," *Computers & Education*, vol. 27, no. 2, pp. 71 – 83, 1996.

[10] H. Kang and P. J. Guo, "Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations," in *User Interface Software and Technology (UIST)*, 2017, pp. 737–745.

[11] S. Kalyuga, P. Ayres, P. Chandler, and J. Sweller, "The expertise reversal effect," *Educational Psychologist*, vol. 38, no. 1, pp. 23–31, 2003.

[12] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?: Nier track," in *2011 33rd International Conference on Software Engineering (ICSE)*, May 2011, pp. 804–807.