Java as a second language:

Thoughts on a linguistically informed transition to typing languages

Eileen King
IB Design
Lakes International Language Academy
Forest Lake, MN
eking@lakesinternational.org

***Abstract*—As use of blocks-based languages as an introduction to programming has become popular, a need has become increasingly apparent for research in how to facilitate the transition from these environments to industry-standard languages. Mediated transfer strategies have been used that are not subject specific; parallels between learning a second human language and a second programming language may provide a more specific framework from which to facilitate the shift. In particular, the noticing hypothesis suggests merit in deliberately pointing out differences between the two, and an experiment is suggested to test how best to implement this idea.**

Keywords—blocks-based; typing; teaching; transfer

I. Introduction

Scratch Day celebrations are on the rise; coding puzzle apps abound; and the Hour of Code has introduced more girls to computer programming in the last two years than in several decades before it combined. Scratch, AppInventor, Blockly, Tynker, and Hopscotch have all arisen to provide an introduction to algorithmic and computational thinking for students who haven't mastered typing or would be put off by the unforgiving syntax of an industry-standard language. The question students and teachers are increasingly asking, then, is what comes next. How best can teachers facilitate the transition to a typing language?

II. Experiment Proposal

A. Motivation

Research into this problem has been informed primarily by what is known generally about the process of transfer and has focused on mediating transfer of programming languages either by implementing the same algorithm in two languages or creating an environment in which students can switch back and forth between two languages, or some combination of these approaches [1][2][3]. However, programming languages are just that -- languages. While they have been carefully and artificially constructed rather than having evolved naturally, they certainly still have grammars and other features shared with human languages. Considering language transfer not just generally but from a linguistic standpoint might provide new insights: we may not be able to raise children as "native" speakers of programming languages, but now that a growing pool exists of students who have a blocks-based first programming language, possibilities exist for borrowing from what we know about second language acquisition (SLA) of human languages. Linguist Richard Schmidt's "noticing hypothesis," for example, proposes that cognitive comparison between one's first and second language (L1 and L2) is a powerful catalyst in language acquisition. Debate is ongoing, however, about how explicitly those comparisons should be made; what I propose here is an investigation of the question with regard to programming languages.

B. Design

I suggest a controlled experiment with three groups: one taught the terminology for comparing their blocks-based L1 to typing L2 while learning L1 (e.g. "These blocks that tell something to repeat over and over are called loops"), one taught L1 with no attempt to foreshadow an L2 but having them explicitly connected in the teaching of L2 (e.g. "This is a while loop; it works the same way that the repeat until block worked in Scratch"), and one taught both languages in succession but left to make the connections on their own. The latter seems illogical, particularly given the work of Dann et al. on mediated transfer [1], but resembles the immersion approach to human language learning (in which use of L1 is effective forbidden), and might serve as a useful control.

Students' success in making the transition could be measured in any number of ways: self-evaluation of proficiency and/or comfort; dropout rates for typing-language classes or programs; and/or teacher analysis of students' output and process.[1] It may also be interesting to borrow again from human language acquisition and think about students' ability to adhere to style conventions of the typing language: that is, do students develop an "accent" from their blocks-based language, and which teaching method, if any, is the most useful in minimizing it? To provide an example, blocks-based Scratch currently includes the ability to declare

[1] Ideally, one of these dimensions should be selected rather than attempting all of them at once; it may be that one approach leads to better student comprehension but another correlates with higher student confidence, for example. Which is most interesting, and what constitutes a successful transition, likely depends on educational philosophy.

variables but no way to specify their type. To determine whether variable *levelComplete* is true, then*, if levelComplete = true* is not only neither redundant nor poor style, as it would be in Java, but the only way to accomplish the check. This sort of construction is common among beginning Java students in general, but would having Scratch background make it harder to learn to write *if levelComplete* instead?

III. Other Considerations

A. Student Interest

This proposal ignores altogether the question of *who* needs to transition from blocks-based to typing languages in the first place. As the answer is likely to rest more in philosophy of computer science education – that is, *why* it's important that students learn to code, and which non-programming aspects of CS are useful for all learners – rather than experimental data, this was deliberate. However, it is important to bear in mind; whether students have self-selected to make the transition or are having it imposed on them is likely to have a dramatic effect on their motivation, which is well known to affect learning.

B. Student Age

Human language teaching points at the fact that students' age is likely to matter enormously in which approach is most beneficial - one of the most concrete differences between child and adult learners of a second language is that young learners maintain their ability to absorb and internalize grammar, whereas adult learners are much more capable of metacognition and gain more from being taught rules explicitly. In keeping with this, it seems reasonable to hypothesize that the first of the three above approaches would have much more to offer older learners than young ones. If that is indeed the case, when in the developmental process does that shift occur? And furthermore, do programming languages, like human languages, have a "critical period" in which a child should be exposed to one in order to become fully proficient later in life? Much research remains to be done.

References

[1] W. Dann, D. Cosgrove, D. Slater, D. Culyba, S. Cooper, "Mediated transfer: Alice 3 to Java." Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12). New York: ACM, 2012, pp. 141-146.

[2] Y. Matsuzawa, T. Ohata, M. Sugiura, and S. Sakai. "Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment." Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). New York: ACM, 2015, pp. 185-190.

[3] M. Armoni, O. Meerbaum-Salant, and Mo. Ben-Ari. "From Scratch to "Real" Programming." Trans. Comput. Educ. 14, 4, Article 25 (February 2015).