Preliminary evidence for learning good coding style with Autostyle

ANTARES CHEN, ELIANE STAMPFER WIESE, HEZHENG YIN, ROHAN CHOUDHURY, AND ARMANDO FOX, UNIVERSITY OF CALIFORNIA, BERKELEY

ABSTRACT

MOOCs for computer science have adopted autograders and hint generators to help students write working code. However, in collaborative, professional settings, code must not only work but also have good style. Good style tastefully uses a language's built-in features and idioms: it is concise, extensible, and communicates the programmer's intent. AutoStyle (Choudhury, Yin & Fox, 2016) makes style feedback scalable by replacing manual code review with automatically-generated hints (based on mining previous student submissions). Our initial findings indicate that AutoStyle can teach a novel coding idiom not taught by instructors, that students can correctly apply the idiom outside the tutor and that it is harder for students to correct working but stylistically-poor code than it is to start from scratch.

AutoStyle uses the widely-used assignment-branch-complexity metric (Fitzpatrick, 2000) as a proxy for good style. AutoStyle provides hints based on a student's current approach, identifying areas for improvement and suggesting language idioms to incrementally improve style. Through interactive practice and feedback, AutoStyle aims to help students learn a three-step iterative process: 1) Search for opportunities for style improvement, especially blocks of code that could be replaced with built-in functions, 2) Find constructs and language idioms by exploring documentation, language guides, and online forums, 3) Correctly apply the new technique.

An initial pilot, with random assignment, found that 70% of the AutoStyle students achieved the best style solution to a Python coding assignment, compared to 13% in the ecological control (Choudhury, Yin & Fox, 2016). That study indicated that AutoStyle improves students' performance while they work with the tutor. The current study (single condition) uses assessments outside the tutor to measure learning. Part 1 (N = 145) replicated the initial finding with a Ruby programming task, with 63% of students achieving the best style solution with AutoStyle. The best solution for Part 1 required a particular Ruby reduction operation, group_by. 47% of students used group_by on their first attempt, while 86% used it on their last attempt. In Part 2, students were given a new programming task, without AutoStyle, where the best style solution would again use group_by. Of the students who returned for Part 2 (N=68), 33% used group_by on their first attempt and, without hints, 61% used group_by on their last attempt. These students were not instructed on group_by between the two parts of the study, suggesting that they learned to use this idiom through AutoStyle's feedback, and they could successfully apply that knowledge to a new task outside the tutor.

Further, students in Part 2 were randomly assigned to solve the problem from scratch, or to begin with a working but stylistically poor solution. On their final submissions, 48% of the students with starter code used group_by, compared to 70% of students who started without code (p=.02, Fisher exact test on usage by condition). Tree-edit distances show that students with starter code made minor tweaks rather than reevaluating the approach. This study suggests AutoStyle's promise for learning, and indicates that starter code may provide nuance to style assessments.

REFERENCES

Choudhury, R. R., Yin, H., & Fox, A. (2016). Scale-driven automatic hint generation for coding style. To appear in the Proceedings of the 13th International Conference on Intelligent Tutoring Systems

Fitzpatrick, J. (2000). Applying the ABC metric to C, C++, and Java. In R.C. Martin, (Ed.) More C++ Gems (245–264). New York, NY: Cambridge University Press.