QIS: Automated Refactoring for Scratch

Peeratham Techapalokul and Eli Tilevich
Software Innovations Lab
Dept. of Computer Science, Virginia Tech
{tpeera4, tilevich}@cs.vt.edu

***Abstract*—Recent studies have demonstrated that the code quality of Scratch projects impacts this learning environment's educational effectiveness. For example, novice programmers are less willing to remix and continue modifying those projects whose code quality is low. This showpiece demonstrates *QIS* (pronounced as /chēz/), a novice-friendly refactoring tool for Scratch 3.0. Integrated with the latest Scratch environment, *QIS* analyzes on-the-fly the code quality of an edited project, displaying the refactorable quality problems as actionable improvement hints. Programmers can then decide to act by carrying out the suggested refactoring to improve the quality of the project's code. *QIS* not only empowers novice programmers to easily and effectively improve their code, but also educates them about the benefits and importance of code quality and its improvement practices.**

I. Introduction

The extraordinary success of Scratch as an introductory computing environment [1] stems not only from its novice-friendly block-based programming interface, but also from its immense global learning community[1], in which community members continuously learn from and build off each others' projects. Nevertheless, recent systematic studies indicate that poor code quality can negatively impact the educational effectiveness of Scratch. Poor quality projects are hard to understand and modify [2], thus stifling the motivation of introductory learners to continue developing increasingly complex and useful projects over time. Furthermore, the communal learning value of a Scratch project diminishes, as it becomes increasingly difficult for others to understand and remix [3]. Recent studies confirm that code quality problems in the Scratch codebase are highly prevalent [4], [5].

The Lehman's Law of Declining Software Quality states that software systems evolve into a state of disrepair unless maintaining their quality extends into a concerted effort over time [6]. Applying this law to Scratch entails that the quality of Scratch projects would continue deteriorating, in the absence of a concerted quality improvement effort. Hence, to prevent the Scratch codebase's quality from deteriorating over time, we introduce *QIS* (**Q**uality **I**mprovement for **S**cratch). *QIS* (plural of qi): in Chinese philosophy and medicine, `qi` is vital energy, whose *balance* is believed to be essential for good health and well-being. By analogy, *QIS* seeks to *balance* the form (i.e., code quality) and function of Scratch code. *QIS*'s beginner-friendly user interface empowers novice programmers to improve the code quality of a worked-on project.

[1]Scratch has close to 41 million users and 42 million shared projects https://scratch.mit.edu/statistics/ (accessed June 2019)

978-1-7281-0810-0/19/$31.00

Software refactoring is a semantics-preserving program transformation that typically improves code quality. Having become part and parcel of modern software development practices, refactoring is a standard feature of modern IDEs for a variety of languages. Thus far, block-based programming has been a notable omission. Indeed, programming environments for blocks lack automated refactoring support, with the exception for the most rudimentary refactoring: renaming of program elements (e.g., variables). Conversely, our experiments have identified 4 Scratch-specific refactorings (3 of them eliminate code duplication and the other reduces variable scope) that are highly applicable and as such can be valuable for Scratch programmers to improve the code quality of their programs [7]. *QIS* renders these refactorings accessible to all Scratch programmers, thus enabling and encouraging them to start improving code quality, so as to avoid the prevalence of recurring quality problems in the Scratch codebase as per prior findings [3].

II. *QIS*: Quality Improvement for Scratch

QIS tightly integrates visual hints that inform programmers about potential quality problems and the automated refactorings that can address these problems. The design of *QIS* is inspired in part by code quality analyzers in IDEs for text-based languages, such as SonarLint[2]—an IDE extension that helps programmer detect and fix code quality issues. Our code smell analysis infrastructure builds on QualityHound, a web-based smell analyzer for Scratch [8], presented as a VL/HCC 2017 showpiece. *QIS* is a research prototype, originally developed to use in the user study evaluation for the paper, accepted for presentation in the main technical program of the VL/HCC 2019 conference. The preliminary findings show that *QIS* can effectively help novice programmers to identify code improvement opportunities and motivate them to engage in improving code quality [7].

QIS accommodates novice programmers in performing the following two cognitively demanding tasks, particularly for beginners: (1) *identifying* refactorable code smells and (2) *transforming* the code to remove the identified code smells. To render *QIS*'s user interface amenable to the target audience of novice programmers, the following design principles guide our user interface design: *1) Code smells as improvement opportunities:* Frame a detected smell as an improvement hint to help programmers identify code smells while maintaining a positive

[2]https://www.sonarlint.org/

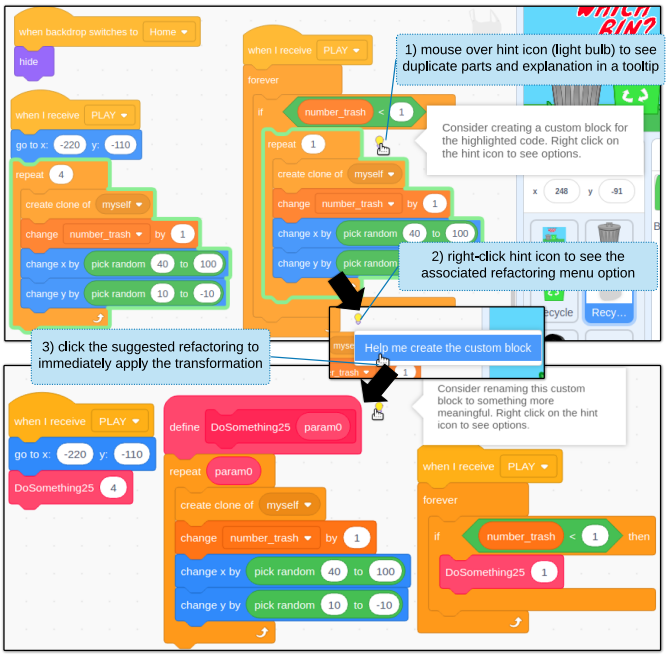

Fig. 1: *Duplicate Code* hint and *Extract Custom Block* refactoring

outlook, so as to avoid the negativity typically associated with highlighting quality problems. *2) Refactoring should be immediately actionable:* Associate refactoring suggestions and precomputed transformations with the presented hint to allow the programmer to immediately apply the suggested refactoring. This design principle favors simplicity over versatility. That is, it would be quite unrealistic to expect that someone new to programming would be capable of selecting the correct refactoring targets and providing the necessary parameters.

QIS performs analysis in the background to provide on-the-fly improvement hints and refactoring suggestions. Its server-side refactoring engine powers the front-end component, which enhances the Scratch programming environment that runs in a Web browser. The front-end refactoring facility periodically sends the edited program to the server and present the analysis results to the programmer in a form of hints and the associated refactorings.

Fig.1 shows an example of how *QIS* presents the detected DUPLICATE CODE smell as a quality improvement hint and the suggested EXTRACT CUSTOM BLOCK refactoring. *QIS* detects the DUPLICATE CODE smell when two or more code fragments containing more than one statement, share an identical structure except for small variations in identifiers and literals. The suggested refactoring is EXTRACT CUSTOM BLOCK; it transforms the program to put the repeated sequences of statement blocks into a new custom block definition script, whose invocations replace the repeated sequences.

The aforementioned design principles have inspired certain implementation choices that we demonstrate by example. Our improvement hints are contextualized using code highlights or abstract visual representations of non-visual program elements to intuitively convey to programmers what and where the problem is. When describing a quality problem, *QIS* makes use of positive language to reframe the problem as a quality improvement opportunity. Finally, *QIS* refrains from asking programmers to specify refactoring parameters (e.g., when a refactoring requires a name, *QIS* uses a placeholder name, followed by another hint that suggests an opportunity to meaningfully rename the placeholder).

III. RELEVANCE

We expect that presenting *QIS* would generate an interesting discussion in the VL/HCC community, as it raises a largely unexplored question: How can programming environments encourage the adherence of beginners to principled software engineering practices? This showpiece demonstrates how a popular mainstream programming environment can be seamlessly enhanced with an automated refactoring facility, as a use case of rendering automated refactoring accessible to programming environments for novices. We seek this opportunity to present our tool as an avenue to receive feedback from fellow researchers and explore opportunities for collaboration.

AVAILABILITY

Our custom Scratch programming editor with *QIS* is available at: https://q4blocks.appspot.com/editor. While our backend refactoring engine is capable of detecting and refactoring all four smells, our front-end user interface currently supports the presentation of the improvement hint for the DUPLICATE CODE smell and the associated EXTRACT CUSTOM BLOCK refactoring. We plan to roll out the support for the remaining hints and their associated refactorings by the time of the showcase.

ACKNOWLEDGEMENTS

This research is supported by the National Science Foundation through the Grant DUE-1712131.

REFERENCES

[1] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[2] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? A controlled experiment on Scratch programs," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, May 2016, pp. 1–10.

[3] P. Techapalokul and E. Tilevich, "Understanding recurring quality problems and their impact on code sharing in block-based software," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2017, pp. 43–51.

[4] G. Robles, J. Moreno-León, E. Aivaloglou, and F. Hermans, "Software clones in Scratch projects: On the presence of copy-and-paste in computational thinking learning," in *Software Clones (IWSC), 2017 IEEE 11th International Workshop on*. IEEE, 2017, pp. 1–7.

[5] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the Scratch repository," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 2016, pp. 53–61.

[6] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, Sep. 1980.

[7] P. Techapalokul and E. Tilevich, "Code quality improvement for all: Automated refactoring for Scratch," in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2019.

[8] ——, "Quality hound an online code smell analyzer for Scratch programs," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2017, pp. 337–338.