# *Quality Hound* — an Online Code Smell Analyzer for Scratch Programs

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

***Abstract*—In this showpiece, we demonstrate the functionality of *Quality Hound* — an online program analysis tool that takes as input a Scratch project and presents to the user a visual representation of the detected quality problems. Made accessible via a browser-based user interface, *Quality Hound* is instantaneously accessible to any Scratch user all over the world. The design of *Quality Hound* is informed by our research on cataloging and automatically detecting recurring quality problems, commonly referred to as *code smells*. We envision that *Quality Hound* can benefit the entire Scratch community by raising the awareness of software quality in this visual programming domain.**

## I. Introduction

Block-based programming has proven spectacularly successful as an educational tool for introductory computing learners. The Scratch community encompasses more than 19.6 million users who have collectively contributed more than 23.7 million projects to a publicly shared online repository[1]. Because of the exploratory and freewheeling nature of Scratch, one may think of *software quality* being at best irrelevant and at worst inimical for the intended key goals of this programming community. If the goal of Scratch is to encourage students to learn how to program, would drawing their attention to the quality of their yet inchoate programming output be detrimental to achieving this goal? Why not postpone all software quality discussion until students have mastered the introductory computing concepts?

However, recent research has shown that the issue of software quality is paramount to block-based programming and the educational objectives that this domain aims to accomplish. Several recent research studies [1], including our own work [2], have shown that Scratch programs are rife with recurring quality problems, which indeed hinder the achieving of various educational objectives. Specifically, because of the communal nature of Scratch, students tend to learn from each other. In that light, poorly designed projects are known to be hard to understand, modify, and reuse. As we have discovered, existing projects that have been heavily evolved by others are likely to exhibit higher quality than those ones left unmodified [2].

In addition, the intuition of *bad habits being hard to break* similarly applies to learning programming practices. As we have discovered, once introductory students develop poor programming habits, they tend to continue following them, even as their level of programming proficiency increases [3]. Hence, the issue of software quality is important and timely in this visual programming domain.

To improve software quality, one must first be able to understand exactly where and how the quality is lacking. The software engineering community has embraced the concept of *code smells* [4] as a shared vocabulary to communicate about recurring quality problems. In the block-based programming community, researchers thus far have made few of their quality analysis tools available to end users for learning and experimentation. Our work is inspired by Dr. Scratch[2] [5], a browser-based tool for assessing Scratch programming proficiency that also detects three code smells (i.e., sprite naming, duplicated code and dead code) in the analyzed code. Dr. Scratch presents the detected smells textually, making it challenging for programmers to trace the identified smells back to the corresponding visual program parts.

In contrast, our tool—*Quality Hound*—improves the utility of software quality analysis for block-based software by placing at the user's fingertips a visual, browser-based code smell analyzer for Scratch projects. The analysis engine of *Quality Hound* comprises the 12 state-of-the-art quality analyzers that we originally developed to produce the experimental results for the paper, appearing in the main technical program of the VL/HCC 2017 conference.

Specifically, *Quality Hound* parses JSON-based ASTs of Scratch programs into an internal representation used by the analysis routines. The analyzers are implemented using JastAdd [6], a Java-based language processing framework. The detected code smells are expressed as block regions, so the smelly blocks or scripts can then be reported to the user. Finally, the presentation layer formats the detected blocks and scripts, to be rendered in the browser by a third-party JavaScript library[3]. Table I briefly summarizes the code smells *Quality Hound* detects. We refer the interested reader to our technical paper [2] for details about these code smells, their prevalence, and their impact on code reuse.

[1] https://scratch.mit.edu/statistics/ (accessed July 2017)

[2] http://www.drscratch.org/

[3] https://github.com/scratchblocks/scratchblocks

978-1-5386-0443-4/17/$31.00 

| Code smell | Description |
|---|---|
| Broad Variable Scope | A variable with its scope broader than its usage |
| Duplicated String | Same strings or substrings are used in multiple places |
| Duplicated Code | Repeated sequence of blocks is used as a way to reuse code |
| Feature Envy | A data producing script is in a different sprite from the one that uses the data, requiring global variable to be used to pass the data around |
| Inappropriate Intimacy | Heavy usages of sensor blocks to check the attribute value of other sprites |
| Long Script | A long script makes code hard to understand |
| Middle Man | A long chain of broadcast receive to delegate work |
| No-Op | A user event-based script that performs no actions |
| Uncommunicative Name | Naming of sprite (started with "Sprite") is not meaningful |
| Undefined Block | A custom undefined block. |
| Unreachable Code | An unreachable event-based script due to the absence of a corresponding broadcast block |
| Unused Variable | A variable is created but remains unused in the project |

TABLE I: Code smells detected by *Quality Hound*

*Quality Hound* renders our analysis infrastructure available to end users, who are expected to possess little to no expertise in software quality analysis. The browser-based user interface lowers the learning curve and increases usability. The tool takes the URL of a Scratch project as input and then presents the detected smells visually to the user in a web browser window. In Figure 1, one can see *Quality Hound* in action and a screenshot of the tool showing the detected smells. Users with some familiarity of the analyzed project can leverage the tool's visual output to navigate to the exact locations of the detected smells. By better understanding why the tool signaled the presence of code smells, programmers can determine if they should engage in quality improvement.

## II. Relevance

*Quality Hound* will be ultimately relevant and interesting to the VL/HCC community because of the multifaceted *visual* aspect of this tool. That is, *Quality Hound* is a *visual* program analysis tools for a *visual* language. It demonstrates how program analysis and quality checking infrastructure can be provided and made accessible for visual languages. Because of the current trend of *democratizing* program analysis and verification tools, so as to make them easily accessible for rank-and-file programmers, our work on designing and implementing *Quality Hound* can offer valuable insights for infrastructure developers of other visual languages.

## III. Conclusion

This showpiece presents *Quality Hound*, an automated program analysis tool for end-users. It enables end-user quality assessment for block-based software. Presenting our tool is expected to spur constructive discussion about various issues related to software quality in block-based software, end-user software quality analysis, and the role of visual tools in pedagogical pursuits.

## Availability

*Quality Hound* can be accessed at: http://research.cs.vt.edu/quality4blocks/projects/quality-hound/.


## Acknowledgements

This research is supported in part by the National Science Foundation through the Grant DUE-1712131 and the Royal Thai Government scholarship.


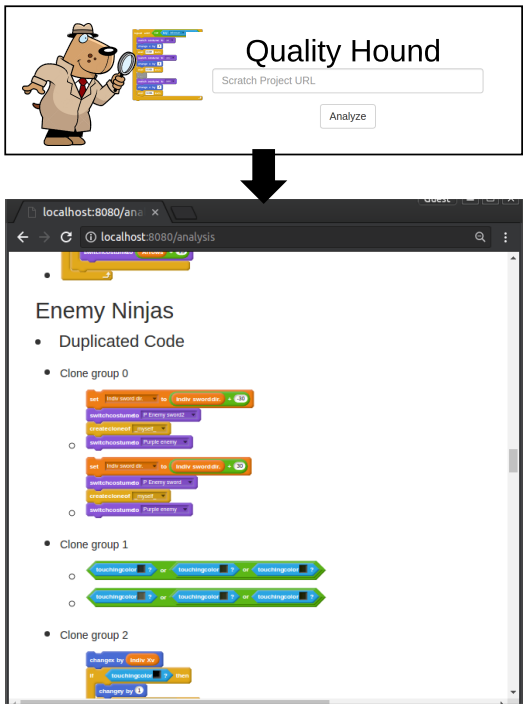


Fig. 1: A screenshot showing detected code smell instances visually presented to the user

## References


[1] 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*, ser. ICER '16. New York, NY, USA: ACM, 2016, pp. 53–61.

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

[3] ——, "Understanding recurring software quality problems of novice programmers," Virginia Tech, Article, 2017. [Online]. Available: http://hdl.handle.net/10919/78337

[4] M. Fowler and K. Beck, *Refactoring: Improving the design of existing code*. Addison-Wesley Professional, 1999.

[5] J. Moreno-León, G. Robles, and M. Román-González, "Dr. Scratch: Automatic analysis of Scratch projects to assess and foster computational thinking," *RED. Revista de Educación a Distancia*, no. 46, pp. 1–23, 2015.

[6] G. Hedin and E. Magnusson, "JastAdd—an aspect-oriented compiler construction system," *Science of Computer Programming*, vol. 47, no. 1, pp. 37–58, 2003.