Reusing in the Small: Promoting Procedural Abstraction in Scratch Communal Learning

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

***Abstract*—One of the most important concepts for budding programmers to master is procedural abstraction. Defined as placing coherent, possibly reusable, functionalities within their own encapsulated program units (e.g., procedures, functions, methods, custom blocks), this concept requires a tangible increase in programming proficiency to master. First and foremost, procedural abstraction is a means of conquering complexity—the ability to convincingly divide the program functionality into distinct coherent parts that are easy to understand and use. In addition, as invocable units of functionality, procedures provide ready-made components that encapsulate the hidden details of their implementation. One of the biggest advantages of Scratch is its seamless support for communal learning, realized as the ability to share and remix projects with ease. However, programmers remix Scratch projects in their entirety, carrying out the corresponding reuse and extension activities at the project level of granularity. In this position paper, we argue in favor of extending Scratch with the ability to reuse individual procedures, implemented as custom blocks. This ability has the great potential benefit of instilling the value of procedural abstraction in the minds of beginner programmers. We discuss how a reuse facility for custom blocks can be added to Scratch by identifying the corresponding design objectives, challenges, and opportunities.**

***Index Terms*—block-based programming, procedural abstraction, reusability, code quality, Scratch, introductory computing**

I. Introduction

One of the most exciting features of major educational block-based programming platforms is their natural support for communal learning, typically referred to as *remixing* and implemented in both Scratch [1] and MIT App Inventor [2], among others. With remixing, a programmer derives a new project by using an existing project as the starting point. In Scratch, for example, one can remix someone else's project with a single mouse click. The environment's remixing system attributes remixes back to their original projects and tracks the total number of remixes.

Remixing has been shown to serve as an effective pathway for novice programmers to learn new programming concepts and constructs [3]. In a way, sharing and remixing projects creates a communal learning environment for the members of community to learn from each other. By motivating programmers to create appealing and useful projects others would want to remix, the Scratch remixing facility fosters social engagement. Finally, remixing bears similarity to forking, a fundamental process in open-source software development, thus preparing beginner programmers for the prevailing programming practices of the wider software development world.

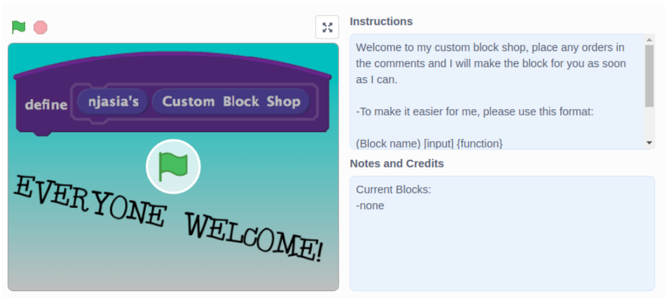
However, remixing projects is not the only approach to software reuse. Oftentimes, programmers incorporate small and modular code components (e.g., procedures), with clearly described functionality, into their projects. This reusable code snippets may come from their own projects or those created and shared by others. Procedural abstraction is an important programming concept, natively supported by major block-based programming languages. If given the ability to reuse procedures, programmers would not only receive a valuable development aid, but also would be conditioned to embrace procedural abstraction as a way to promote modular decomposition, encapsulation, and reuse. Nevertheless, major block-based programming lack any facilities for sharing and reusing existing procedures.

In this position paper, we motivate the need to add such facilities and also argue that that the required engineering effort would be quite manageable. Specifically, we give an overview of the current support for code reuse in Scratch, the potential benefits and educational opportunities that a code reuse facility can bring, as well as the ideas and considerations for implementing such a facility.

II. Reusing Code in Scratch

Scratch programmers are limited to only reusing the code they have encountered previously as a result of interacting with their own code or recollecting their exposure to relevant functionality while browsing third-party projects. A programmer opens the project containing the sought-after functionality, locates the relevant code, copies it into a clipboard-like storage (e.g., "backpack" in Scratch), and finally pastes it into the project that needs the code.

Reusing only what a programmer can recall is somewhat limited given the vast codebase the Scratch community has at its disposal. Furthermore, based on our observations, we discovered the Scratch community extending a workaround effort that reflects the need to share and reuse code. Specifically, some Scratch programmers create special projects that contain only custom blocks; such projects typically follow the naming convention of "* custom block shop," with a concrete example shown in Figure 1. Some Scratch programmers even commit to curating their custom block projects by performing



Instructions

Welcome to my custom block shop, place any orders in the comments and I will make the block for you as soon as I can.

-To make it easier for me, please use this format:

(Block name) [input] {function}

Notes and Credits

Current Blocks:
-none

Fig. 1. A custom block shop project

community-building activities that include accepting third-party contributions and opening their curated custom block collections for others to reuse.

One facility that can be very helpful in reusing code is the ability to search code by name, a feature absent from major block-based programming environments, despite their large communal codebases. For Scratch, one can add a minimal but very helpful facility for indexing and searching for procedures. Currently, only Scratch projects are searchable by name.

III. Potential Benefits of Supporting Code Reuse

Enhancing programming environments for blocks with the ability to share and reuse code would provide various productivity and educational benefits, as we discuss next.

Improving Productivity and Creativity Systematic support for reusing code (e.g., in a form of a function library) can greatly increase programmer productivity. Without a rich ecosystem of libraries and frameworks, building any non-trivial software project would become a massive undertaking for the average programmer. The ability to reuse code systematically via libraries and frameworks provides reliable and useful building blocks, whose underlying implementation details remain hidden. On the other hand, Scratch programmers are left to their own devices to dissect the code they may want to use. Lack of modularity and high complexity can quickly make this code foraging task beyond reach for novice programmers. The required difficult trial-and-error process can not only decrease programmer productivity, but also cause them to become less creative. While tinkering to reuse existing code can be a valuable educational experience in many cases, a facility for reusing common and well-defined functionalities can help programmers stay in their creative flow. From a pedagogical standpoint, code reuse facilities would provide an authentic learning experience that familiarizes students with a fundamental, real-world software development practice.

Serving as Programming Examples Once pedagogical concern is that when novice programmers can search and reuse code with ease, they may miss out on some educational benefits, such as understanding third-party code and learning in the process. This fear of unintended consequences is unwarranted, as programmers reuse ready-made programming solutions as a regular practice. If a found solution accomplishes the task at hand, then programmers feel compelled to learn how that solution works. We believe that the ability to search for existing solutions in Scratch will afford similar benefits to this programming community. Small and modular code snippets would serve as programming examples, a valuable educational asset for novice programmers. Both remixing [3] and code reuse [4] have been observed as influencing beginner programmers to use increasingly more programming constructs.

Promoting Code Quality Building on the success of remixing, code reuse can make a positive impact on the educational effectiveness of Scratch as a learning environment for introductory computing. For one, by using this facility, programmers would end up improving the code quality of their projects (i.e., *non-functional* quality). In other words, systematic code reuse can remove some of this domain's most prevalent recurring quality problems (e.g., *Code Duplication*, *Long Script*, etc.), as identified previously [5].

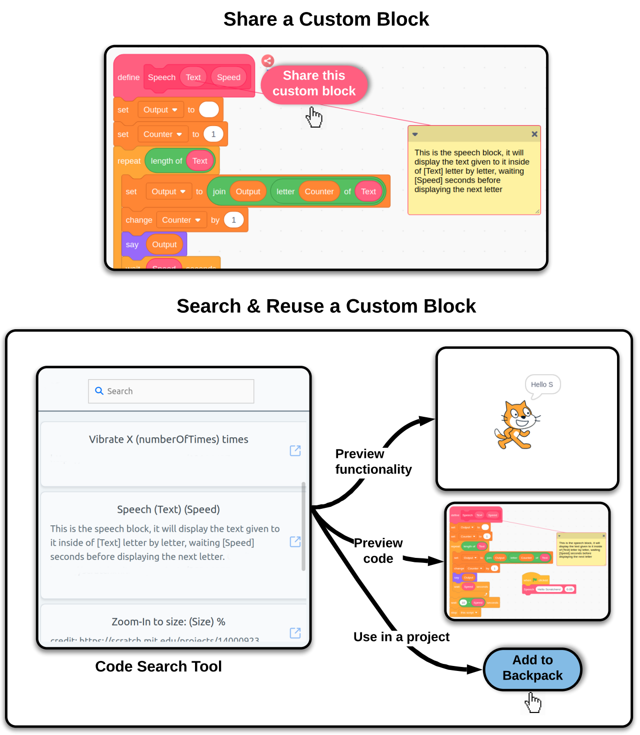
The block-based programming community can promote procedural abstraction as a way to encourage those programmers with sufficient programming proficiency to start thinking about their code quality. Besides procedural abstraction and parameterization, we expect programmers to develop various programming competencies, which arise from the need to share useful code (i.e., identifying reusable code) and write code that can be found and reused by others (i.e., coming up with a descriptive name).

A code reuse facility can increase the overall project's modularity, an important factor influencing whether a project is conducive to remixing. Prior studies have identified the low usage of the procedural abstraction in block-based programming [6], [7], despite the concept being quite important to both introductory CS and disciplined software engineering practices. A lack of modular decomposition may negatively affect the communal learning value of a Scratch project by making it hard for others to understand, reuse, and modify. Indeed, as pointed out by the designer of the Scratch remixing system, a project's modularity is one of many factors that increase the chance of a project being remixed by others [1].

IV. Code Reuse Facilities for Scratch

We discuss some of the main technical implications of adding a code reuse facility to Scratch and other programming environments for blocks. Figure 2 outlines the user interface of a code reuse facility. This mock user interface shows how the key functionalities—*share* and *search & reuse*—can be exposed to the programmer.

Sharing Custom Blocks Certain Scratch design choices hinder writing self-contained reusable code. For example, in the absence of local variables, to reuse the code of some custom blocks would require introducing new sprites or global variables. Additionally, with custom blocks lacking return values, a separate external variable needs to be introduced to access the result computed by a custom block. Perhaps it is because of these language design choices that some Scratch projects contain explanatory comments that explain the intended application of the aforementioned workarounds. Nevertheless, a dedicated facility for reusing code is expected to encourage programmers to expend additional efforts making



This is the speech block, it will display the text given to it inside of [Text] letter by letter, waiting [Speed] seconds before displaying the next letter

This is the speech block, it will display the text given to it inside of [Text] letter by letter, waiting [Speed] seconds before displaying the next letter.

credit: https://scratch.mit.edu/projects/14000923

Fig. 2. Basic code reuse facilities to support programmers to share and reuse custom blocks

their code easier to understand and reuse (e.g., describing what a custom block does and how others can use it).

Before programmers can be encouraged and supported to share custom blocks, they would first need to start using custom blocks in their code. Given that this language feature is underused in the codebase of block-based languages [6], there is great potential benefit in lowering the programming burden of extracting reusable custom blocks from extant projects. Hence, automated refactoring in this domain can serve yet another valuable purpose [8].

Reusing Custom Blocks It would not be hard to enhance Scratch with a facility for reusing custom blocks, as the language already provides basic bookkeeping of variables, automatically creating non-existing variables used in the copy-and-paste code. Furthermore, the insights and findings of the prior state of the art in the code reuse practices of novice programmers (e.g., Looking Glass [9]) would inform the user interface design for a Scratch facility for reusing code. A code search facility can be integrated with automated program analyses to be able to meaningfully rank search results. The ranking could take into account various ease-of-reuse characteristics of the shared custom blocks when searching them based on their name and description.

Finally, to motivate code reuse, the proposed facilities would have to incorporate the open-source culture of the remixing practice. To that end, the ability to track the usage of shared custom blocks would increase their visibility, while also helping programmers identify when custom blocks are worth reusing. The authors of shared custom blocks should be properly attributed to receive a due credit for their work being used by others. For example, to add a third-party custom block in a project, a code reuse facility can add an automatically generated code comment, associated with the block's definition script, ID, author, and description. This author credit information can also be extracted and included as part of the project description.

V. Conclusion

The success of Scratch in fostering the open-source culture, in which programmers share and remix projects, is quite unprecedented for novices and end-user programmers. Indeed, as the Scratch community continues to grow, it currently amasses almost 42 million users and 43 million shared projects[1], Code reuse facilities could become a valuable addition to the remixing facility, fulfilling the common programming need to reuse existing code. Custom blocks—a basic programming construct for procedural abstraction in Scratch—presents a viable opportunity to reuse code at smaller levels of granularity. Code reuse facilities could provide various development and educational benefits that range from improving programmer productivity to promoting code quality, a subject of future controlled user studies that would drive further design iterations.

Acknowledgment

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

References

[1] A. Monroy-Hernandez, "Designing for remixing: Supporting an online community of amateur creators," Ph.D. dissertation, Massachusetts Institute of Technology, 2012.
[2] S. C. Pokress and J. J. D. Veiga, "MIT App Inventor: Enabling personal mobile computing," *arXiv preprint arXiv:1310.2830*, 2013.
[3] S. Dasgupta, W. Hale, A. Monroy-Hernández, and B. M. Hill, "Remixing as a pathway to computational thinking," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, ser. CSCW '16. New York, NY, USA: ACM, 2016, pp. 1438–1449. [Online]. Available: http://doi.acm.org/10.1145/2818048.2819984
[4] P. Gross and C. Kelleher, "The Looking Glass IDE for learning computer programming through storytelling and history exploration: conference workshop," *Journal of Computing Sciences in Colleges*, vol. 26, no. 1, pp. 75–76, 2010.
[5] P. Techapalokul and E. Tilevich, "Understanding recurring quality problems and their impact on code sharing in block-based software," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2017.
[6] 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.
[7] I. Li, F. Turbak, and E. Mustafaraj, "Calls of the wild: Exploring procedural abstraction in App Inventor," in *2017 IEEE Blocks and Beyond Workshop (B&B)*, Oct 2017, pp. 79–86.
[8] 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.
[9] P. A. Gross, M. S. Herstand, J. W. Hodges, and C. L. Kelleher, "A code reuse interface for non-programmer middle school students," in *Proceedings of the 15th international conference on Intelligent user interfaces*. ACM, 2010, pp. 219–228.

[1]https://scratch.mit.edu/statistics/