Enabling End-Users to Implement Larger Block-Based Programs

Nico Ritschel
ritschel@cs.ubc.ca
University of British Columbia
Vancouver, Canada

Felipe Fronchetti
fronchettl@vcu.edu
Virginia Commonwealth University
Richmond, United States

Reid Holmes
rtholmes@cs.ubc.ca
University of British Columbia
Vancouver, Canada

Ronald Garcia
rxg@cs.ubc.ca
University of British Columbia
Vancouver, Canada

David C. Shepherd
shepherdd@vcu.edu
Virginia Commonwealth University
Richmond, United States

ABSTRACT

Block-based programming, already popular in computer science education, has been successfully used to make programming accessible to end-users in applied domains such as the field of robotics. Most prior work in these domains has examined smaller programs that are usually simple and fit a single screen. However, real block-based programs often grow larger and, because end-users are unlikely to break them down into separate functions, they often become unwieldy. In our study, we introduce a function-centric block-based environment to help end-users write programs that require a large number of blocks. Through a user study with 92 users, we evaluated our approach and found that while users could successfully complete smaller tasks with and without our approach, they were both quicker and more successful with our function-centric method when tackling larger tasks. This work demonstrates that adding scaffolding can encourage the systematic use of functions, enabling end-users to write larger programs with block-based programming environments, which can contribute to the solution of more complex tasks in applied domains.

ACM Reference Format:
Nico Ritschel, Felipe Fronchetti, Reid Holmes, Ronald Garcia, and David C. Shepherd. 2022. Enabling End-Users to Implement Larger Block-Based Programs. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3510454.3528644

1 PROBLEM OUTLINE

Programming has become an integral part of the work of millions of employees. However, as statistics of the US Department of Labor [13] show, most of these employees are not professional developers. They are *end-user programmers* who have not generally received formal education or training for performing programming-related tasks. Because end-user programmers have little programming experience and knowledge, they need tools that are specifically designed with their needs in mind [4, 17]. The most popular form of beginner-friendly programming tools are *block-based programming environments*. Block-based environments combine simplified high-level programming languages with a user interface that uses visual blocks to represent program syntax and a drag and drop interface to enable the blocks to be intuitively composed. These mechanisms allow blocks to avoid many of the the frustrations that beginner programmers traditionally experience [9].

While block-based environments were originally conceived for computer science education [9], they have been demonstrated to support end-user programmers in other domains. For example, end-users have automated their homes [6], developed augmented reality apps [10], and programmed industrial robots [16] using block-based environments. Previous evaluations of end-users creating block-based programs found them to be easy to learn and use [6, 10, 14, 15], but these evaluations were almost exclusively based on small programs that fit on a single screen. Code that fits on a single screen without scrolling is easier to understand; linting and coding styles have long been used to restrict the length of coding units in professional software development [1, 3, 5]. Thus, to write a longer program that is still understandable, developers decompose their program into short, related units of functionality (i.e., functions), which requires experience and expertise that end-users may not typically have.

Although most block-based environments provide procedural abstractions, end-users rarely structure their programs using these abstractions. Instead, they create programs that exhibit properties commonly understood to be problematic, like long methods and code clones [8, 11]. While block-based languages support abstraction and decomposition in theory, they do little to encourage their use, and so end-users do not decompose their programs, even though decomposition is crucial when creating larger programs. This work addresses this issue and presents a domain-specific block-based programming environment that makes program decomposition accessible to end-users in the field of robotics. Inspired by previous work on block-based industrial robot programming [16], we chose the programming of *mobile robots* as our use case, as these robots are commonly used in warehouses and in small manufacturing facilities and could potentially be programmed by end-users [12]. Although the mobile robot tasks we chose were conceptually simple—moving boxes from station to station or re-arranging a stack of boxes—successful completion required programs larger than a single screen (50+ blocks), which would naturally be better solved using functions.

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05.
https://doi.org/10.1145/3510454.3528644

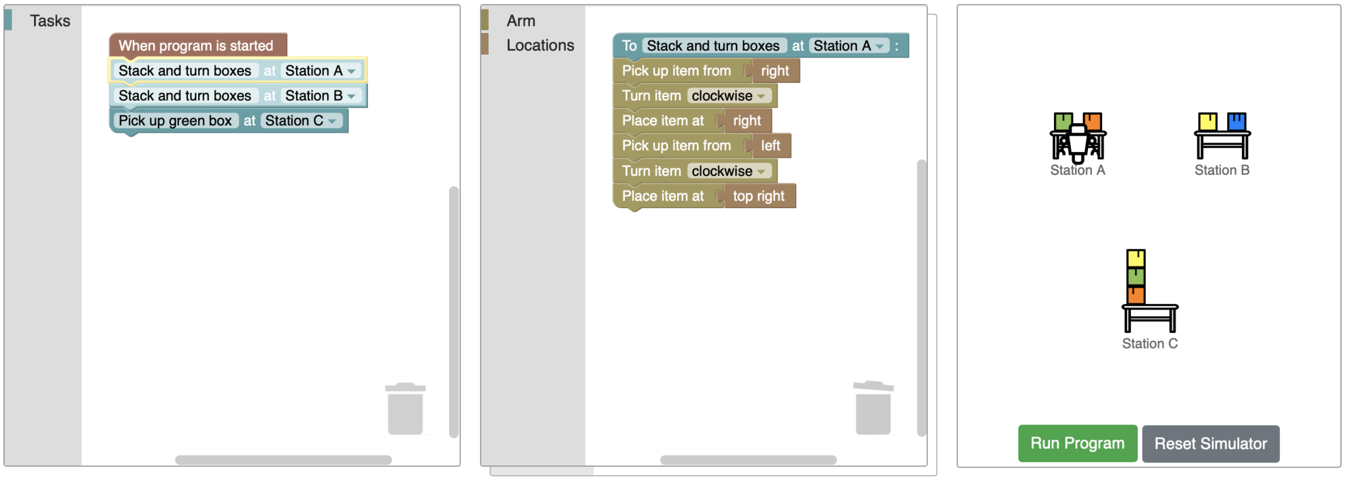

Figure 1: Proposed scaffolded programming system with side-by-side canvases. The left canvas shows the main program while the right one shows the body of the currently selected task(s). Users can test programs using the simulator on the right side.

2 APPROACH OUTLINE

We propose a programming system for mobile robots that supports functions in a *scaffolded* style. We believe that domain-specific scaffolding can provide explicit support for users as they apply functional decomposition to their program. Figure 1 shows our system, which features a strict one-level hierarchy of domain-specific functions, which we call *tasks*. Tasks are always assigned to a single, localized workstation and only contain commands that the robot can carry out within the scope of that workstation. Our system further supports this hierarchical design with an environment that splits the programming environment into side-by-side canvases, where one provides the user with an overview of their program and the other shows the content of the currently selected task.

Our hypothesis is that this scaffolded approach to program decomposition and code navigation offers end-users a simplified, yet powerful way to structure their programs. We have used the framework of *13 Cognitive Dimensions of Notation (CDN)*[1] [7], which provides terminology for analyzing visual languages, to motivate and analyze the presented design. Most notably, the presented approach to function scaffolding allows end-users to program without having to plan and decide on the structure of their programs in advance. This reduces the amount of Premature Commitment that end-users face, as well the Error-proneness of manual re-structuring of code in the absence automated refactoring tools as they are offered by professional development environments. The system's side-by-side canvases support the overall design by improving the Visibility of task-specific code and separating it from the high-level program structure. Unlike previous approaches [2], this structure does not require the end-users to perform the Hard Mental Operation of manually setting up and maintaining such a hierarchy.

3 EVALUATION AND RESULTS

To gather insights on how end-users structure their programs in traditional block-based environments, and to compare our approach to the traditional way of programming, we conducted a randomized online experiment[2]. We recruited 92 end-user participants via *Amazon Mechanical Turk (AMT)* and randomly assigned each participant to use either our scaffolded programming environment or a similar, traditional block-based programming environment. We asked participants to complete three interactive tutorials that taught them how to use the respective systems and how to use functions and tasks to structure their programs. We then asked participants to complete a series of three tasks with increasing difficulty to determine if and how they could solve each task. The first task was small enough to fit on a single screen while the other two were longer. The second task was substantially easier to identify program structure than the third one.

In line with previous studies, we found that end-users from both cohorts performed well on the first task that fit on a single screen. For the second task, we found that 60% of those participants that used a traditional programming environment did structure their programs, and 69% of those chose a structure that was aligned with our scaffolded approach. For the final task, only 28% of the participants using a traditional environment structured their programs, with no clearly dominating style present. Despite a median length of 65 blocks per program, the remaining participants wrote their entire program as a single, linear sequence of commands.

Overall, participants who structured their programs performed substantially better. Those participants who used the scaffolded environment that required them to structure their programs had a 20% higher success rate in the second task and a 29% higher success rate for the third task. We therefore conclude that the scaffolded environment helped participants significantly when they solved larger programming tasks.

4 ACKNOWLEDGEMENT

This work was funded by the NSF under grant NRI-2024561.

[1] CDN terminology is underlined in the remainder of this section.

[2] A complete version of our experiment is available online at: https://researcher2021.github.io/mobile-robots/

REFERENCES

[1] Marwen Abbes, Foutse Khomh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2011. An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension. In *2011 15th European Conference on Software Maintenance and Reengineering*. 181–190.

[2] Andrew Bragdon, Robert Zeleznik, Steven P Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J LaViola Jr. 2010. Code bubbles: a working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2503–2512.

[3] Sofia Charalampidou, Apostolos Ampatzoglou, and Paris Avgeriou. 2015. Size and cohesion metrics as indicators of the long method bad smell: An empirical study. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. 1–10.

[4] Brian James Dorn. 2010. *A case-based approach for supporting the informal computing education of end-user programmers*. Ph.D. Dissertation. Georgia Institute of Technology.

[5] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, Amsterdam.

[6] Mateus Carvalho Gonçalves, Otávio Neves Lara, Raphael Winckler de Bettio, and André Pimenta Freire. 2021. End-user development of smart home rules using block-based programming: a comparative usability evaluation with programmers and non-programmers. *Behaviour & Information Technology* (2021), 1–23.

[7] Thomas R. G. Green and Marian Petre. 1996. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing* 7, 2 (1996), 131–174.

[8] Felienne Hermans, Kathryn T Stolee, and David Hoepelman. 2016. Smells in block-based programming languages. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 68–72.

[9] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[10] José Miguel Mota, Iván Ruiz-Rube, Juan Manuel Dodero, and Inmaculada Arnedillo-Sánchez. 2018. Augmented reality mobile app development for all. *Computers & Electrical Engineering* 65 (2018), 250–260.

[11] Gregorio Robles, Jesús Moreno-León, Efthimia Aivaloglou, and Felienne Hermans. 2017. Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning. In *International Workshop on Software Clones (IWSC)*. 1–7.

[12] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. 2011. *Introduction to autonomous mobile robots*. MIT press.

[13] US Department of Labor. 2021. Occupational outlook handbook. *https://www.bls.gov/ooh/* (2021).

[14] David Weintrop. 2019. Block-based programming in computer science education. *Commun. ACM* 62, 8 (2019), 22–25.

[15] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*. 1–12.

[16] David Weintrop, David C Shepherd, Patrick Francis, and Diana Franklin. 2017. Blockly goes to work: Block-based programming for industrial robots. In *Proceedings of the Blocks and Beyond Workshop (B&B)*. 29–36.

[17] Susan Wiedenbeck, Patti L Zila, and Daniel S McConnell. 1995. End-user training: an empirical study comparing on-line practice methods. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 74–81.