Scaffolded Block-based Instructional Tool for Linear Data Structures

A Constructivist Design to Ease Data Structures' Understanding

https://doi.org/10.3991/ijet.v14i10.10051

Daniel Felipe Almanza-Cortés, Manuel Felipe Del Toro-Salazar,
Ricardo Andrés Urrego-Arias
Universidad El Bosque, Bogotá D.C., Colombia

Pedro Guillermo Feijóo-García (✉)
University of Florida, Gainesville, FL, USA
Universidad El Bosque, Bogotá D.C., Colombia
pfeijoogarcia@ufl.edu

Fernando De la Rosa-Rosero
Universidad de los Andes, Bogotá D.C., Colombia

Abstract—Data structures courses commonly introduce topics involving high levels of abstraction and complexity, requiring significant effort from instructors and students to achieve positive teaching-learning outcomes. Despite the multiple studies that have occurred within the Computer Science Education (CS ED) community to understand the experiences that novice programmers may have when learning how to program, there is still a lack of exploration and research on understanding these experiences in scenarios different from first-year Computer Science (CS) courses. Looking further from CS introductory courses, this paper presents the results of a pilot study that evaluated the interaction of a group of CS Colombian students with *DStBlocks,* which is a scaffolded block-based instructional technology created to ease linear data structures understanding. The findings and results of this pilot study are favorable, corresponding to tests centered on user experience and learning impact.

Keywords—Data structures, blocks-based language, visual blocks programming, instructional technologies, scaffolding

1 Introduction

The Computer Science Education (CS ED) community exposes interest on how students learn how to program. There have been multiple initiatives, projects, and studies centered on favoring learning processes novice programmers may experience when learning how to program, with a general focus on understanding first-year Computer Science (CS) introductory courses (e.g. CS0, CS1, and CS2). Likewise, the

CS ED community has reported several initiatives and projects on designing scaffolded educational technologies to help novice programmers to learn programming and algorithmics. Contexts on storytelling, art, and game designing, are considered with the purpose of engaging the learner with meaningful experiences. Nevertheless, literature on upper-level courses is still infant, and contexts like data structures courses (CS3) are still unexplored [1, 2].

Despite CS3 is a mandatory course in CS curricula, and a course generally taught after introductory programming courses, learners struggle when dealing with its topics and their high-level of abstraction. This is due to the complexity regarding time and memory management, and the programming languages and paradigms used for the application of the course's concepts [2, 3].

Arguments to understand why learners struggle in this course may come from different directions. Struggling might happen because on the pedagogical strategies designed and followed within the course, or on how we present the topics and the micro-curricula generally structured for it: linear, hierarchical, and nonlinear data structures. Likewise, it may also be due to a possible lack of connectivity between concepts, skills, and mental models built in previous courses, referring to what CS3 requires, demands, and expects from the learners.

With CS3 as a blue sea to explore, and looking to find how we may enhance learning processes for this course, we designed and conducted this study to analyze how algorithmic visualization and the use of interactive block-based languages could ease linear-data-structures learning. The latter, designing and evaluating a new scaffolded tool: *DStBlocks*, which allows learners to explore interactively the algorithmics corresponding to the design and development of linear data structures using a block-based language, avoiding the need to face syntax problems of a particular programming language.

This paper presents findings of the interaction of a group of Colombian undergraduate students, from two different institutions (Universidad El Bosque and Universidad de los Andes), with the *DStBlocks* tool. Data for this study was gathered quantitatively and qualitatively, evaluating the usability, acceptance, and pedagogic impact.

This document is organized as follows: section 2 presents the theoretical framework, enouncing theories, definitions, previous works and existing tools related to the context of interest. Section 3 presents the software design and architecture behind *DStBlocks*, complemented by its design concept description. In section 4, the methodology of the experimental phase is explained, following up with section 5, which presents our findings and results. The final part of this article presents the study's conclusions and a proposed future research related to this project.

2 Theoretical Framework

In this section, we present the theoretical framework, the pedagogical concepts, the models and taxonomies used for this study, as well as previous work related to the topic.

2.1 Constructivism

Constructivism, proposed by Jean Piaget in 1952, is an educational model that considers the relationship between the learner, his/her social environment, his/her experiences, and the interaction among them [4]. It conceives learning not as a result of predefined structures or operations, but as an outcome from the mental models that the learner constructs and re-constructs while interacting with his/her environment and relating past to present experiences. Therefore, it is essential for a constructivist strategy to understand the learner and his/her vital field; the theory refers to this as the learner's meta-cognitive understanding of his/her environment, formed by his/her past, present and future [5, 6].

Unlike other models, like behaviorism and transmissionism, constructivism considers the learning process to be learner-centered [5, 6] and it is based on how the learner's interaction with his/her context provides learning outcomes through the assimilation of experiences from the surrounding environment's understanding [6, 7]. According to this model, it is the learner's duty to design and define the structure of what he or she will identify as "knowledge", based on the experiences and information provided by the surrounding environment, and the connections between this information, and experiences coming from the past and the present [6, 7].

2.2 Bloom's taxonomy

Proposed in 1950 by Benjamin Bloom, this taxonomy explores the different ways in which knowledge is acquired and retrieved [8]. The taxonomy is recognized by categorizing and adequately ordering different thought skills to obtain cognitive mastery [8]. These thought skills are described as nouns and they are: evaluation, synthesis, analysis, application, understanding, and knowledge. Following this specific order, the first skills are known as Higher Order Thinking Skills (HOTS), while the latter ones are conceived as Lower Order Thinking Skills (LOTS) [8]. Bloom's Taxonomy proposes to work from the LOTS towards the HOTS, considering that to evaluate, the learner needs first to be able to synthesize, which comes after the ability to analyze and apply knowledge that has already been understood [8].

In 2000, Anderson and Krathwohl proposed a revision to the original Bloom's Taxonomy, conceiving the thought skills according to the practices designed and followed within the classroom [8]. Different from the taxonomy's original design, this revised version defines the skills not as nouns but as verbs, also changing the order of the considered skills, giving priority to the synthesis over the evaluation [8]. The thought skills according to this revised version are: create, evaluate, analyze, apply, understand, and remember [8].

A more recent version of Bloom's Taxonomy was proposed in 2008, considering the Information and Communication Technologies (ICTs) as part of nowadays teaching-learning processes [8]. Churches justifies this digital Bloom's Taxonomy on how ICTs are present in educators' and learners' daily lives, proposing to prioritize the creation-skill over the rest, arguing on how ICTs provide environments in which a learner can achieve other thought skills while creating or working hands-on [8]. The

thought skills, according to this digital version, are: create, remember, understand, apply, analyze, and evaluate [8].

2.3 Scaffolding

Scaffolding refers to the assistance the instructor may provide to the learner to guide his/her learning process to succeed. This guidance may be structured and given in different ways according to the experience the learner expresses [9], helping him/her to achieve better thinking and problem-solving competencies [10].

An example of a scaffolding strategy is what the *GoalPlanCode Editor (GPC)* proposes. GPC Editor was designed to support students' learning on elementary Pascal programming using a "tight process control" scaffolding approach [9]. The tool asked the students to decompose a problem into goals and sub-goals, asking them to identify those code fragments (proposed with the tool) that achieved the identified sub-goals [9]. The latter, to proceed composing the fragments together into a final program [9]. This strategy was successful, as the authors describe in their work, because they found that students who moved from the GPC Editor to Think Pascal, were able to write programs that were highly structured [9]. However, in our opinion, the most interesting finding from their work is on how students wished that the scaffolding approach disappeared, after interacting some time with the GPC Editor [9]. Scaffolding strategies have to vary according to the student's learning needs and achievements [10].

On a different approach, MOOSE Crossing provides a different and more motivational scaffolding strategy [11]. This system was designed to help children to learn how to program while building virtual worlds together, as part of an online community [11]. It offers an appealing environment with a text-based adventure game feel, letting the children not just experience the world, but also build it [11]. The children work with it doing creative writing and computer programming while establishing a new relationship to reading, writing, and programming [11]; it focuses on students' motivation as a basis [9].

Similarly, considering what Brandsford et al. [10] expressed in their work, simulations and computer-based models can perfectly enhance the student's skills in a topic. An example of the previous statement is *StarLogo* as a tool, which was designed to help children model and explore the behaviors of decentralized systems [11]. This tool lets students think about the actions and interactions of individual objects, helping them to understand, visualize and connect, through the use of complex systems such as traffic jams and ant colonies [11]. The tool's visual and interactive assistance provides students with a stronger personal connection to the underlying models and, finally, helps them to enhance their performance [11].

These previous examples are presented to better explain what scaffolding is, and how it may be adapted according to a proposed learning context. Considering the definition given above and the philosophy of constructivism, it is the educator's duty to identify what scaffolding strategies may be proposed and applied, according to the learner's needs, skills and achievements.

2.4 Academic background and similar technologies

Feijóo-García and De la Rosa, as presented in [12], performed a study on a tool designed to help Colombian elementary and high school students learn to program using mobile robotics as an educational context. The tool they designed and evaluated, entitled *RoBlock*, provides the learner with scaffolded levels using a block-based language, corresponding to programming-related concepts like variables, sensors, conditionals, cycles, and functions [12]. The evaluation of *RoBlock* addressed learning and usability indexes, measured with pre-assessments and post-assessments centered on how engaging the tool was in accordance with its purpose, and how were the experiences the learners had with it [12]. The results of this study were generally positive, and *RoBlock* was satisfactorily accepted by the students as a ludic, engaging, and user-friendly tool [12]. Moreover, the experimental design followed up by Feijóo-García and De la Rosa in [12], was considered to evaluate *DStBlocks* from both perspectives: usability and learning. The latter, because of the indexes they designed, and because of how their study measured the interactions between the learner and the new technology.

Based on a block-based language as *RoBlock*, *Pencil Code* is a block-based tool proposed and designed to help novice programmers move between block and text coding, building in them, through its use, enough confidence so that they can create programs without needing to use the tool [13]. Although other tools already offer features on translating block code to text (e.g., *Blockly, Snap, BlockEditor*), *Pencil Code* differs from other tools as it translates blocks code to an educational programming language, allowing the learners to transition easily from a rich scaffolded blocks environment to a text one, as they familiarize with syntax while improving their skills [13]. The tool was designed to teach programming to novice learners, of all ages, providing learning contexts involving art, storytelling, and music [13]. Considering a possible lack of assistance between visual blocks programming (VBP) languages/tools to professional programming languages/technologies, *Pencil Code* was designed to help novice programmers bridge between a scaffolded drag-and-drop block-based language to text, introducing idioms used by professional web developers [13]. The latter, guiding the learners to be confident enough to build programs without the use of this tool [13].

Weintrop and Holbert conducted an interesting study to understand how learners use *Pencil Code* to work in either a blocks-based or text-based interface, analyzing what modality learners choose to work in, and the reasons why they move from one representation to the other within a single project [14]. The authors studied two sets of students consisting on 13 high school (HS) girls and six students at a graduate course (GC) on educational learning environments (four women and two men) [14]. The HS set worked three 100-minute classes to create an interactive website to promote the class they were taking, using features regarding images, user-inputs responses, and drawing [14]. The GC students took, first, a survey on previous programming experience, and then had an activity ("create a quilt") that was left as homework for a one-week doing, having all the actions with *Pencil Code* logged by the authors [14]. Weintrop and Holbert concluded that the dual-modality approach is effective for pro-

gramming learning, considering that all the students could successfully complete the assessments [14]. Furthermore, they also concluded that blocks are useful to introduce a new programming environment, as well as support items for conceptual comprehension [14].

Zumaytis and Karnalim designed a tool to help students understand better the Branch & Bound strategy and its characteristics. The *AP-BB* tool included four modules for Brute Force strategy visualization, Branch & Bound strategy visualization, Reduced Cost Matrix (RCM) visualization, and case-based performance comparison [15]. The authors evaluated the tool using a qualitative method with 20 students who answered a set of 13 questions, asking aspects related to the purpose of the tool and its usability [15]. In general, the study reported that the students agreed on how the tool's design fulfilled the user's necessity towards the Branch & Bound strategy, inviting the authors to proceed with further studies on the tool and its curricular impact on the algorithmic strategy course at their university [15].

3 DStBlocks: Solution's Architecture and Design

We designed DStBlocks with a Model-View-Controller (MVC) architecture pattern, as it appears in Figures 1 and 2, using an Object-Relational Mapping (ORM) technology for database access and management.

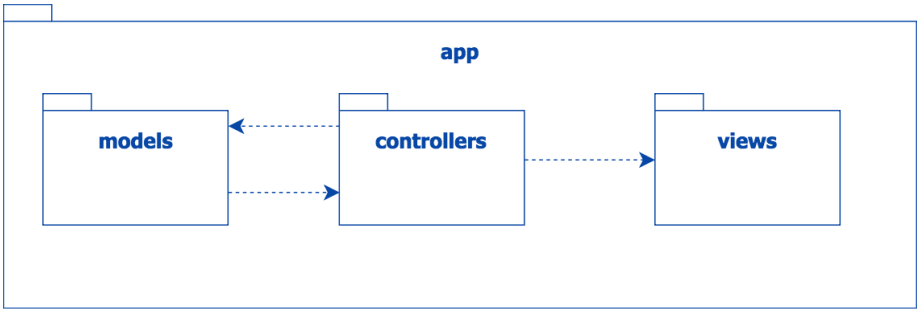

Fig. 1. Package Diagram

We used *Ruby on Rails* Web Framework for general development, *Google Blockly*, to include block-based code features, and *Vis.js*, to graphically represent the linear data structures. As it appears in Figure 2, *DStBlocks* was deployed in the Cloud using *Heroku*.

DStBlocks has a total of ten logical entities, as we illustrate in Figure 3. Students may come from different institutions and have multiple workspaces, one per data structure offered with the tool. Additionally, we modeled the procedure as an individual entity, so that the users get to have access to previous attempts on a data structure, easing continuous practice, feedback, and learning.

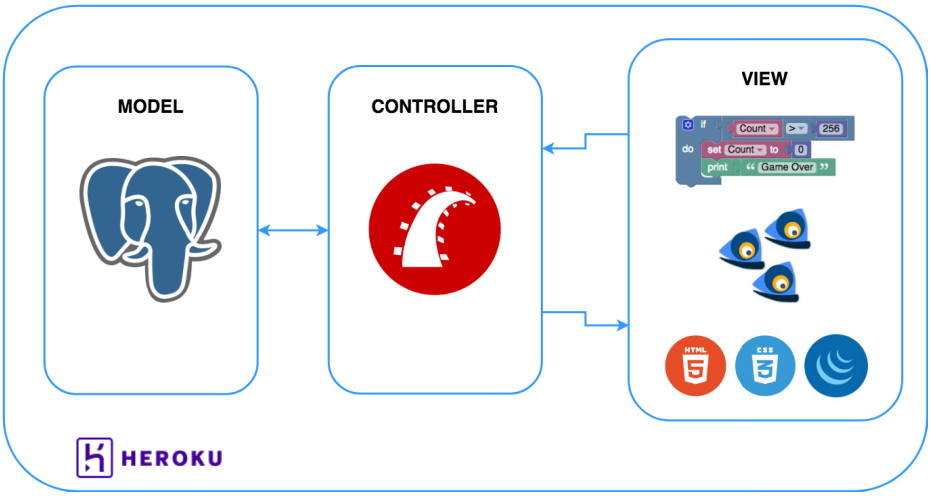

Fig. 2. Overview Diagram

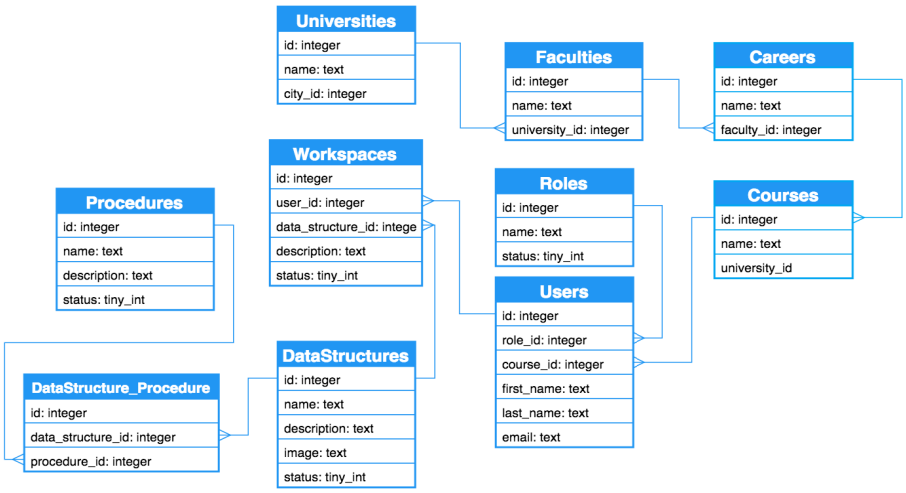

Fig. 3. Data Model Diagram

After following the development and implementation phases, we obtained a robust Web Application that includes CRUD functionalities for four linear data structures: stack, queue, simple linked list, and doubly linked list; in addition to a scaffolded block-based canvas and a dynamic visual representation of the data structure and its behavior. The tool and its features were elaborated in Spanish, according to the context in which we were conducting the study (Bogotá, Colombia) and looking to ease user-interaction for the Hispanic students.

Fig. 4. *DStBlocks* - Login Screen

The interaction with *DStBlocks* begins when the student opens the tool and logs in to have access to all its features (Figure 4). Already authenticated, the student observes the main-menu screen illustrated in Figure 5, in which he/she can either select a previously created data structure or create a new one.

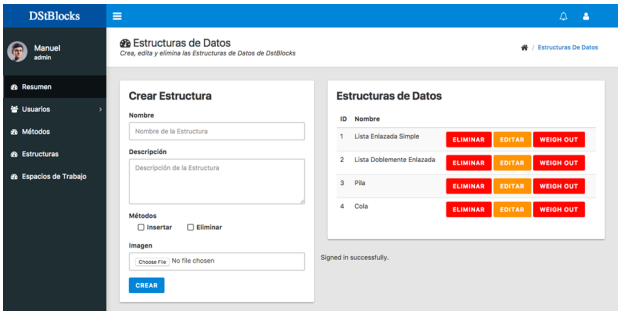

Fig. 5. . *DStBlocks* – Main-Menu Screen

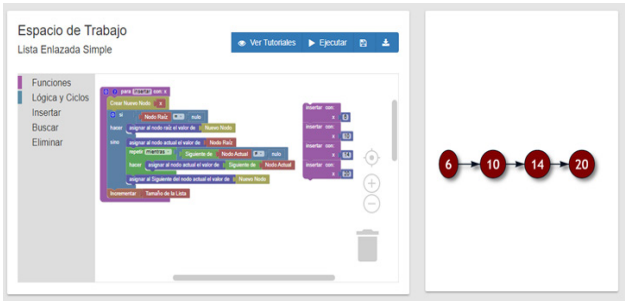

Fig. 6. *DStBlocks* – Workspace Screen (Simple Linked List)

When the student picks a data structure from the main menu screen (Figure 5), he/she is redirected to the workspace screen, which has a code-based scaffolded canvas so that he/she can work on CRUD functions with the data structure selected: insert

element, search element, or delete element (Figure 6). This workspace screen provides visual feedback of his/her work through the block-based canvas and the visual representation of the data structure. Finally, the student has the chance of saving the current workspace, viewing explanatory video tutorials, or exporting the code to supported *Blockly* programming and markup languages: Python, JavaScript, or XML.

4 Experimental Design: Usability and Learning Outcome

To validate *DStBlocks* as a scaffolded block-based instructional tool for linear data structures, we designed an experiment using two different lenses. The first lens was centered on user-experience design, validating the usability of the tool using quantitative and qualitative methods further detailed in [16]. The second lens was proposed to observe how the tool adapted within the learning processes of the study participants, following the method applied in [12]. Both lenses where considered looking to identify design breakdowns and opportunities to enhance learning towards the tool's context. These are explained in the following subsections.

4.1 First phase: usability testing

This phase consisted on a 45–60 min supervised interaction process, in which the learner was invited to use the existing features of the tool while measuring the time required to access and use each one of them. After interacting with the tool, the learner answered two qualitative questions regarding his/her experience with the tool and what he/she considered should be improved or updated to afford a better user experience.

Table 1. Tool's Tested Features

Feature ID	Feature Description
F1	Create a new user account.
F2	Log in to an existing user account.
F3	Create a new workspace.
F4	Edit an existing workspace.
F5	Build the method "insert element" for the stack.
F6	Build the method "remove element" for the stack.
F7	Build the method "insert element" for the queue.
F8	Build the method "remove element" for the queue.
F9	Build the method "insert element" for the simple linked list.
F10	Build the method "remove element" for the simple linked list.
F11	Build the method "insert element" for the doubly linked list.
F12	Build the method "remove element" for the doubly linked list.

For this test, we had the voluntary participation of 26 second-year Computer Science (CS) students from two different institutions.

The students who participated in the study had already successfully completed the data structures course before using the tool for the first time. This was because we

looked to see how their understanding varied while interacting with this tool, and to gather feedback regarding their experiences with its features. The latter, considering their previous experiences with the data structures courses.

4.2 Second phase: learning-outcomes and usability testing

This phase consisted of two in-class assessments and an out-of-class activity with DStBlocks, working exclusively with one data structure: the stack. The out-of-class activity consisted of three open tasks with DStBlocks, centered on the data structure operation and coding with the tool: create a new workspace (data structure), insert an element (push), and remove an element (pop). This activity was proposed after a summative pre-assessment, and followed by a summative post-assessment; both designed with two open questions addressing two different aspects: what and how. The first question, centered on what, invited the students to describe what a stack was, requesting them to provide examples of how they understood the data structure. This question was the same for both assessments, being always the first one asked to the students. Complementarily, the second question requested the learners to work on how the stack operated, requesting them to complete the code of one CRUD operation. For the pre-assessment, they were asked to complete a function to push an integer into a stack of integers (Figure 7), while for the final assessment the function they were asked to complete was to pop an integer from the same stack (Figure 8). This experimental approach was based on [12].

```
public void pushElement(int[] stack, int element)
{
    //Complete Method
}
```

Fig. 7. . Pre-assessment's second question: Complete the Push Method

```
public int popElement(int[] stack)
{
    //Complete Method
}
```

Fig. 8. Post-assessment's second question: Complete the Pop Method

For this phase, we had the voluntary participation of 15 second-semester Computer Science (CS) students, with no previous experience on Data Structures courses. This was because we looked to see how their understanding arose while interacting with this tool, gathering feedback regarding their experiences with its features, and looking to reduce, as much as possible, biases from any potential previous courses. Finally, at the end of this phase, we asked two qualitative questions on usability looking to gather the learners' perception about the tool and its features.

5 Findings and Results

In this section, we present findings and results obtained after applying both tests described in section 4. Our findings and results are presented by test, regarding the lenses used on usability and learning outcome.

5.1 First phase: usability testing

As we can observe in Figure 9, the average time users reported to spend accessing and using the requested features is favorable, considering that the users did not take more than 420 sec, or seven minutes, to properly work and respond to any of the requested operations. This, also recalling that most of the features from F5 to F12 required algorithmic procedures from the students on the linear data structures (i.e. CRUD operations).

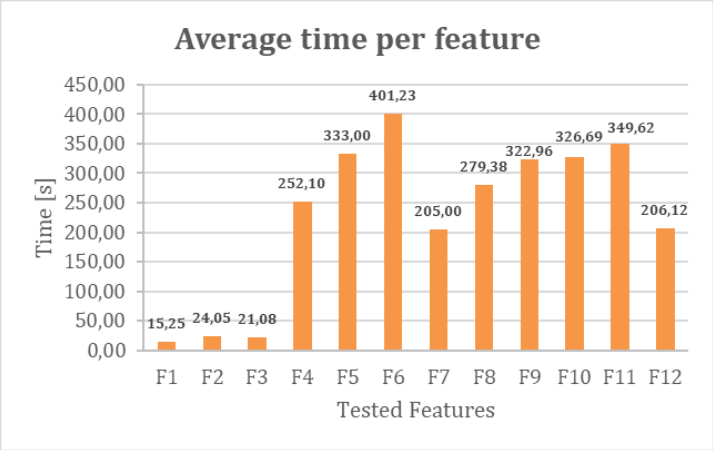

Fig. 9. Usability Test- Average Times per Feature

As shown in Figure 9, the first three features of the linear data structures present a similar behavior regarding data insertion and removal, with greater average times on data removal. The fourth linear data structure, the doubly linked list, presents the opposite behavior; this could be because the students interacted previously with the simple linked list, which presents a similar removal operation. However, the interaction between algorithmic features F5 to F12 does not differ for more than three minutes, which is acceptable according to the study's ranges and considerations.

Figure 10 presents the responses on how appropriate the students consider the tool's graphical proposal and organization. 73% consider it appropriate, and 8% find it very appropriate. This is a positive result, considering that 81% of the students find the tool's visual proposal easy to use and friendly for its purpose.

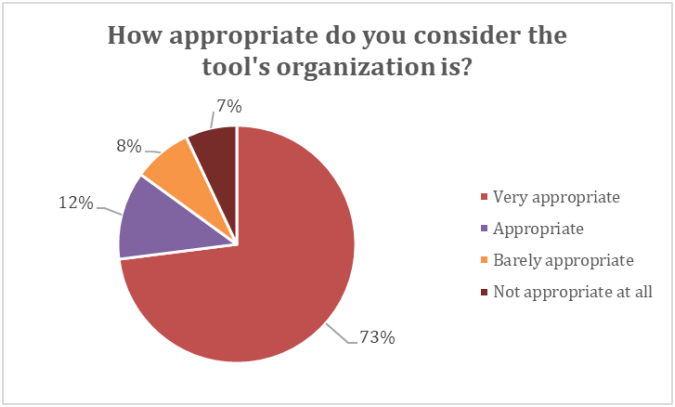

Fig. 10. Usability Test- Tool's Organization Perception

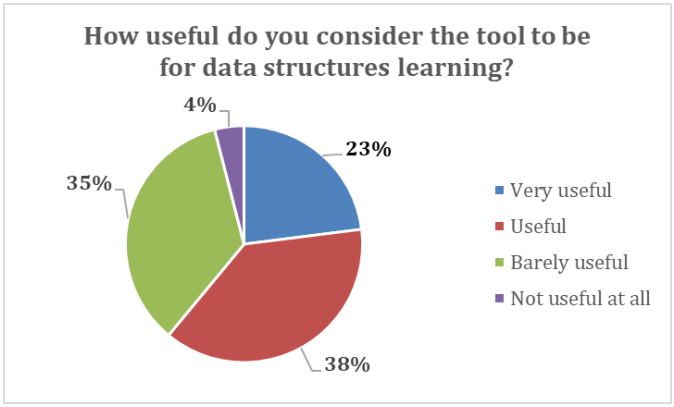

Fig. 11. Usability Test- Tool's Usefulness Perception

Figure 11 presents the students' perceptions regarding how much they consider the tool useful to learn about data structures, according to their already existing experience with a data structures course. The common perception is positive considering that, as we can observe, 23% of the students consider it very useful while 38% consider it useful: this means 61% of the students found the tool useful according to its context.

In general, the test's results show that the tool was well received by the audience that interacted with it. Furthermore, the perception and measured interaction tell us that *DStBlocks* is easy to use and visually friendly to users.

5.2 Second phase: learning-outcomes and usability testing

Regarding the method described in section 4.2, Figure 12 presents students' distribution based on the responses gathered for the first question of the pre-assessment.

The responses were categorized by the research team, considering that this pre-assessment's questions were open-questions.

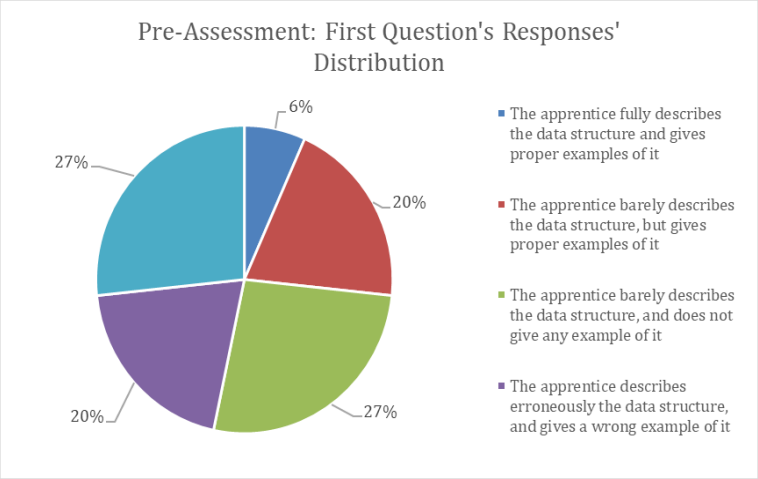

Fig. 12.. Pre-Assessment: First Question – Distribution of Responses

Likewise, Figure 13 presents the students' distribution based on their responses for the same question on the post-assessment, after their interaction with *DStBlocks* for the out-of-class activity described in section 4.2. The same categories used in Figure 12 are considered for Figure 13 and, as we can observe, there was a significant increment on the number of students who were able to describe the Stack, and give examples regarding its operation: from 26% to 100% considering the first two categories in both charts.

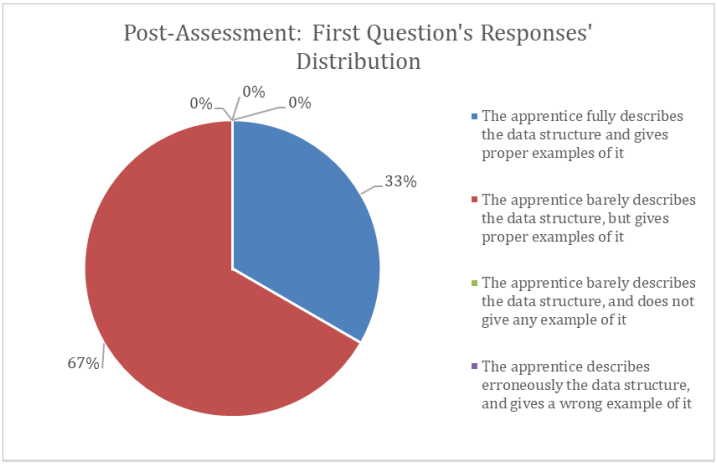

Fig. 13. Post-Assessment: First Question – Distribution of Responses

Referring to the second question of the post-assessment, Figure 14 exposes the students' distribution based on how well they completed the method requested, using Java to build the method (see Figure 7). For this question, three different categories were used, based on the open-responses given by the students.

Similarly, Figure 15 presents the students' distribution based on their responses to the second question of the post-assessment, which also requested them to work using Java. Using the same categories of Figure 14, we see that the proportion of students who could not address or complete the method decreased to 0%, while 93% did it with just syntax errors, and 7% completed it with no errors at all. These are significant results, considering that the students gained understanding on the data structure's operation after interacting with *DStBlocks*, which was something missing at the time of the pre-assessment, and this shows that the tool did help them to understand *what* the data structure was and *how* it is supposed to operate.

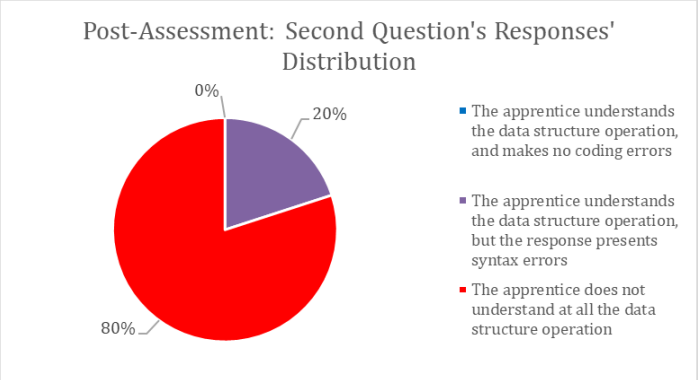

Fig. 14. Post-Assessment: Second Question – Distribution of Responses

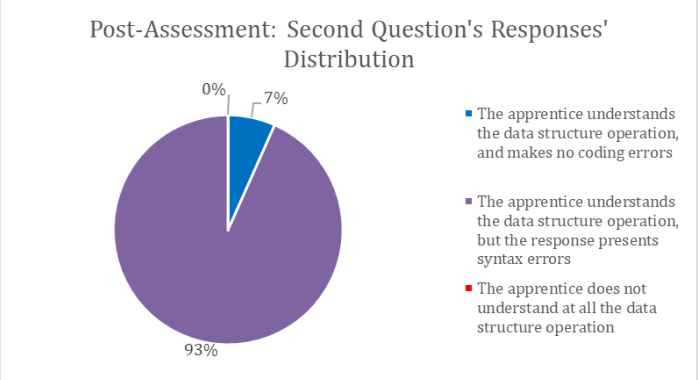

Fig. 15. Post-Assessment: Second Question – Distribution of Responses

At the end of this phase, we asked the students to respond to three questions on how useful they considered the tool was after interacting and learning with it. Figure

16 gathers the students' responses distribution on how intuitive they considered the blocks' menu was. As we can observe, 73% of them found the blocks' menu highly intuitive while the rest indicated it was barely intuitive. Something positive of these responses is that none of them considered the blocks' menu to be not intuitive.

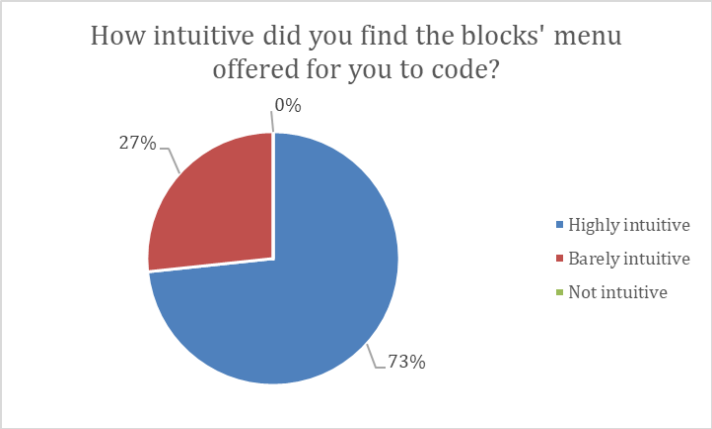

Fig. 16. User Testing – Blocks' Menu

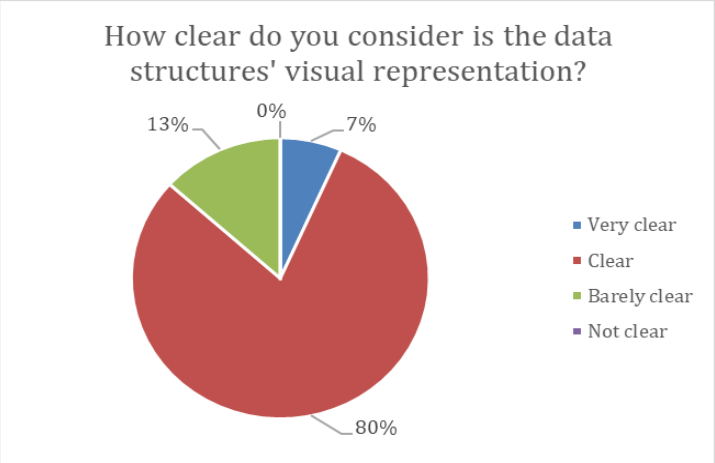

Fig. 17. User Testing – Data Structures' Visual Representation

Similarly, Figure 17 presents the students' distribution based on their responses on how clear they considered the data structures' visual representation was. As Figure 17 illustrates, 87% of the students provided positive responses: 80% found it clear and 7% answered "very clear". These responses are favorable to this first pilot of *DStBlocks*, considering that a significant majority found the data structures' visual design clear, which indicates that the proposed design concept for *DStBlocks* proved to be a user-friendly scaffold.

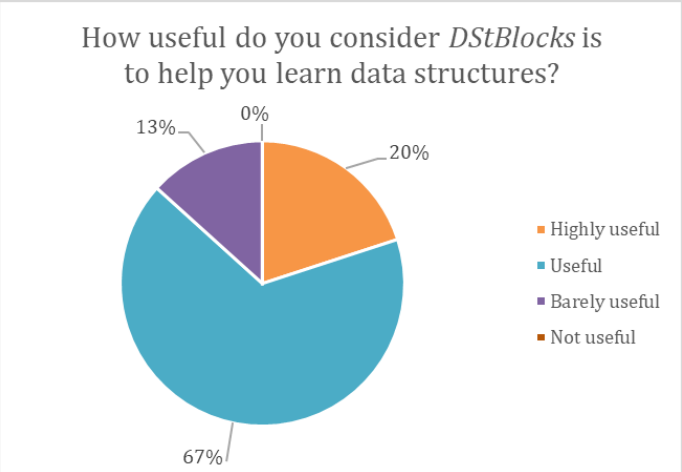

Fig. 18. User Testing – Tool's Usefulness

Finally, Figure 18 gathers the students' distribution based on their responses on how useful they found *DStBlocks* was to learn data structures. As we can observe, 87% of the students answered positively to this question, with 20% of them answering with "highly useful" and 67% indicating that it was "useful". These responses tell us that *DStBlocks* responded favorably to its purpose and that it was well received by the target population, considering that these students interacted with the tool for an out-of-class learning assignment.

6 Conclusion and Future Work

Considering the findings and results we present and describe in section 5, we conclude that the first version of *DStBlocks* was successfully received by the target population, who considered its design concept to be attractive, interesting and user-friendly (please refer to section 5). Moreover, we can conclude that the tool's visual organization and scaffolding help students to learn data structures, fulfilling the main objective of being an application that interactively guides students to learn data structures. Thus, considering the perceptions gathered in the second phase of the experimental method, and the learning outcomes that came with the interaction of the students with the instructional tool.

Following this pilot, there is a wide range of opportunities for future work considering technological, experimental and pedagogical lenses. *DStBlocks* currently assists on linear data structures, so it would be interesting to explore possible extensions aimed to provide scenarios with non-linear data structures such as graphs and trees. Although the same graphical and interactive approach could be used with these tentative new data structures, it could also be possible to extend the design concept to complement this graphical proposal with real-time error tracking, and a robust log system.

Furthermore, there are more possibilities to experiment with *DStBlocks*. For this pilot, the experimental method considered time measurement and students' perceptions to evaluate the tool, after they had interacted with it and its features. Because the tool has been evaluated only with on-activity approaches, possible pivotal strategies may consider in-activity's analysis with mechanisms such as the think-aloud protocol [17], looking to collect learners' experiences while they interact with the tool. This, to evaluate the tool's usability and pedagogic impact in a process-based qualitative method.

It is evident that there are several possibilities to extend the project, considering that the current pilot study successfully shows that *DStBlocks* responds satisfactorily to the purpose for which it was designed. The tool can be extended, as well as the methodology used, by modifying and improving the schema that we proposed and implemented for this first step.

7 Acknowledgement

The authors would like to express their most sincere gratitude to the students who actively participated in this pilot study. Their participation and feedback were essential for this study to succeed.

8 References

[1] Zingaro, D., Taylor, C., Porter, L., Clancy, M., Lee, C., Liao, S. N., & Webb, K. C. (2018). Identifying Student Difficulties with Basic Data Structures. *Proceedings of the 2018 ACM Conference on International Computing Education Research - ICER 18*. https://doi.org/10.1145/3230977.3231005

[2] Begy, V. and Schikuta, E. (2016). A lightweight e-learning system for algorithms and data structures. In Proc. 18th Int. Conf. on Information Integration and Web-Based Applications and Services, Singapore, 199-208. https://doi.org/10.1145/3011141.3011181

[3] Lai, A. and Wu, P. (2015). The development of simulation-based learning system for binary tree of data structures. In Proc. 15th Int. Conf. on Advanced Learning Technologies, Taiwan, 296-298. https://doi.org/10.1109/ICALT.2015.39

[4] Vergara-Rios, G. and Cuentas-Urdaneta, H. (2015). Current Term of Pedagogical Models in the Educational Context. OPCION, 31, pp. 914-934.

[5] Feijóo-García, P. G., & De la Rosa, F. (2018). Instructional Application for Programming and Algorithmic Self-Learning - A Didactic Approach with Mobile Robotics as Pedagogical Context. *Proceedings of the 10th International Conference on Computer Supported Education*. https://doi.org/10.5220/0006690002300237

[6] Ben-Ari, Mordechai. "Constructivism in Computer Science Education." *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education - SIGCSE 98*, 1998, https://doi.org/10.1145/273133.274308

[7] Feijóo-García, P. G., Medina-Cortés, D. F., Ramírez-Cajiao, M. C., & Espinosa-Díaz, E. E. (2017). Cooperative Learning Web Application for Water Care in Colombia – Manglar: Actor-Network Theory Software Solution. *International Journal Of Emerging Technologies In Learning (IJET), 12*(04), pp. 208-216. https://doi.org/10.3991/ijet.v12i04.6733

[8] Churches, A. (2009). Taxonomía de Bloom para la Era Digital. Eduteka Retrieved from: http://eduteka.icesi.edu.co/articulos/TaxonomiaBloomDigital

[9] Soloway, E., Guzdial, M., & Hay, K.E. (1994). Learner-Centered Design: The Challenge for HCI in the 21st Century, *Interactions*, vol. 1, no. 2, pp. 36–48, https://doi.org/10.1145/174809.174813

[10] Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). Chapter 9: Technology to Support Learning. *How people learn: Brain, mind, experience, and school: Expanded Edition*. National Academies Press. pp. 206-230. https://doi.org/10.17226/9853

[11] Resnick, M., Bruckman, A., & Martin, F. (1996). Pianos Not Stereos: Creating Computational Construction Kits, *Interactions*, vol. 3, no. 5, pp. 40–50, https://doi.org/10.1145/234757.234762

[12] Feijóo-García, P., & De la Rosa, F. (2016). RoBlock – Web App for Programming Learning. *International Journal Of Emerging Technologies In Learning (IJET), 11*(12), pp. 45-53. https://doi.org/10.3991/ijet.v11i12.6004

[13] Bau, D., Anthony-Bau, D.A., Dawson, M., & Sydney-Pickens, C. (2015). Pencil code: block code for a text world. In Proceedings of the 14th International Conference on Interaction Design and Children (IDC'15). ACM, pp. 445-448. https://doi.org/10.1145/2771839.2771875

[14] Weintrop, D., & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'17). ACM, pp. 633-638. doi: https://doi.org/10.1145/3017680.3017707

[15] Zumaytis, S., and Karnalim, O. (2017). Introducing an Educational Tool for Learning Branch & Bound Strategy, *Journal of Information Systems Engineering and Business Intelligence*, vol. 3, no. 1, pp. 8-15,https://doi.org/10.20473/jisebi.3.1.8-15

[16] Cruz-Rodríguez, A., Ospina-Gómez, S., & Feijóo-García, P. (2019). GALILEO: A Georeferenced System proposed for Emergency Services' Population Control. *Ingenieria Y Universidad*, *23*(1). https://doi.org/10.11144/Javeriana.iyu23-1.ggsp

[17] Teague, D. & Lister, R. (2014). Programming: reading, writing and reversing. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. (ITiCSE'14). ACM, pp. 285-290. https://doi.org/10.1145/2591708.2591712

9 Authors

Daniel Felipe Almanza-Cortés is a Systems Engineer graduated at Universidad El Bosque, Bogotá D.C., Colombia. (E-mail: dalmanzac@unbosque.edu.co).

Manuel Felipe Del Toro-Salazar is a Systems Engineer graduated at Universidad El Bosque, Bogotá D.C., Colombia. (E-mail: mtoros@unbosque.edu.co).

Ricardo Andrés Urrego-Arias is a Systems Engineering senior student at Universidad El Bosque, Bogotá D.C., Colombia. (E-mail: rurrego@unbosque.edu.co).

Pedro Guillermo Feijóo-García is a Fulbright-Colciencias scholar and a Doctoral Researcher at the Engaging Learning Lab, Computer & Information Science & Engineering Department, University of Florida, Gainesville, FL, U.S.A. Additionally, he is a Core-Faculty Assistant Professor of the Systems Engineering Program, Universidad El Bosque, Bogotá D.C., Colombia. (E-mail: pfeijoogarcia@ufl.edu).

Fernando De la Rosa-Rosero is an Associate Professor of the Systems and Computing Engineering Department, Universidad de los Andes, Bogotá D.C., Colombia. (E-mail: fde@uniandes.edu.co).

Article submitted 2018-12-24. Resubmitted 2019-02-21. Final acceptance 2019-02-21. Final version published as submitted by the authors.