Willow: A Tool for Interactive Programming Visualization to Help in the Data Structures and Algorithms Teaching-Learning Process

Pedro Moraes
phsm@cin.ufpe.br
Federal University of Pernambuco
Recife, Pernambuco

Leopoldo Teixeira
lmt@cin.ufpe.br
Federal University of Pernambuco
Recife, Pernambuco

ABSTRACT

Data Structures and Algorithms (DSA) are one of the main pillars of software development; however, abstractions around them are hard to teach and to be understood by students. The most common approaches adopted by instructors to demonstrate the behavior of DSAs are the use of resources like slides and whiteboard sketches to create program illustrations. This task may be slow and tedious because these illustrations need to be continuously updated to represent new algorithm inputs and modifications. In this paper, we propose Willow , a tool for Program Visualization Simulation (PVS), which supports user interactions to manipulate the generated visualizations. With these manipulations in the visualization, we expect the user to be able to create better examples, resembling Algorithm Visualization Simulation tools (AVS), which are specialized in providing visualizations for specific DSAs. We evaluated our tool through a preliminary qualitative study with teaching assistants from an introductory Computer Science course who all give review lessons to the students. Our preliminary results show that the tool was well accepted by the participants, but we still need more studies to validate the use of the tool in classrooms. With the use of our tool features in the teaching-learning process, we expect that instructors may be able to interactively and more clearly explain DSAs to their students, without the hassle of hours creating slides or drawing by hand messy examples of algorithms.

KEYWORDS

data structures and algorithms, program visualization, algorithm visualization, programming learning

ACM Reference Format:
Pedro Moraes and Leopoldo Teixeira. 2019. Willow: A Tool for Interactive Programming Visualization to Help in the Data Structures and Algorithms Teaching-Learning Process. In *XXXIII Brazilian Symposium on Software Engineering (SBES 2019), September 23–27, 2019, Salvador, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3350768.3351303

SBES 2019, September 23–27, 2019, Salvador, Brazil

ACM ISBN 978-1-4503-7651-8/19/09…$15.00
https://doi.org/10.1145/3350768.3351303

1 INTRODUCTION

Data Structures and Algorithms (DSA) classes in Computer Science and related courses are very challenging for both instructors and students. Students have to understand how a program source code represents abstract ideas related to algorithms. In trying to overcome the difficulty of their students, instructors usually recur to lecture slides or sketching diagrams on whiteboards to show algorithms illustrations [10].

Both slides and sketching have disadvantages. The former requires a great deal of planning and preparation from the instructors to create good quality examples, the latter takes time to draw during the class and the diagrams can get quite messy [17], possibly making referred DSAs even harder to understand. Moreover, none of them are really capable of showing dynamic algorithm visualizations, which is the capability of accepting new inputs or changes and consequently draw new visualizations.

Another approach for showing how DSAs work is through the use of graphical tools, that might be categorized as Program Visualization Simulation (PVS) and Algorithm Visualization Simulation (AVS) tools. Most of these tools are dynamic and allow the instructor and the students to navigate through the program and explore it step by step [3, 26].

Despite the rich history of research works in the program visualization field, graphical tools still have limitations that make them less effective for teaching DSAs. They are either too broad, being capable of generating visualization to many programs, but these are not detailed enough to explain DSAs, or too narrow, providing good visualizations only for a small set of algorithms [8, 10, 22].

To overcome these limitations, we propose a tool called Willow. The innovation aspects that our tool brings is the combination of the best aspects of both PVS and AVS tools. Therefore, our tool is capable of creating visualizations for any provided program. These visualizations can then be customized to create a representation of the data structure or algorithm in the program which favors its understanding.

It is already known that graphical tools are being adopted by some introductory computer science courses (CS1) in prestigious universities [10] and can bring benefits to instructors and students [8]. Introductory computer science and introductory programming are two common terms to reference the initial programming courses in many works. We decided to use the term introductory computer science in this work. Given this scenario, our tool raises some questions that must be answered to validate its use in programming and DSAs learning. We want to investigate what benefits our tool can bring not only to CS1 classes, like other tools did [10, 13], but

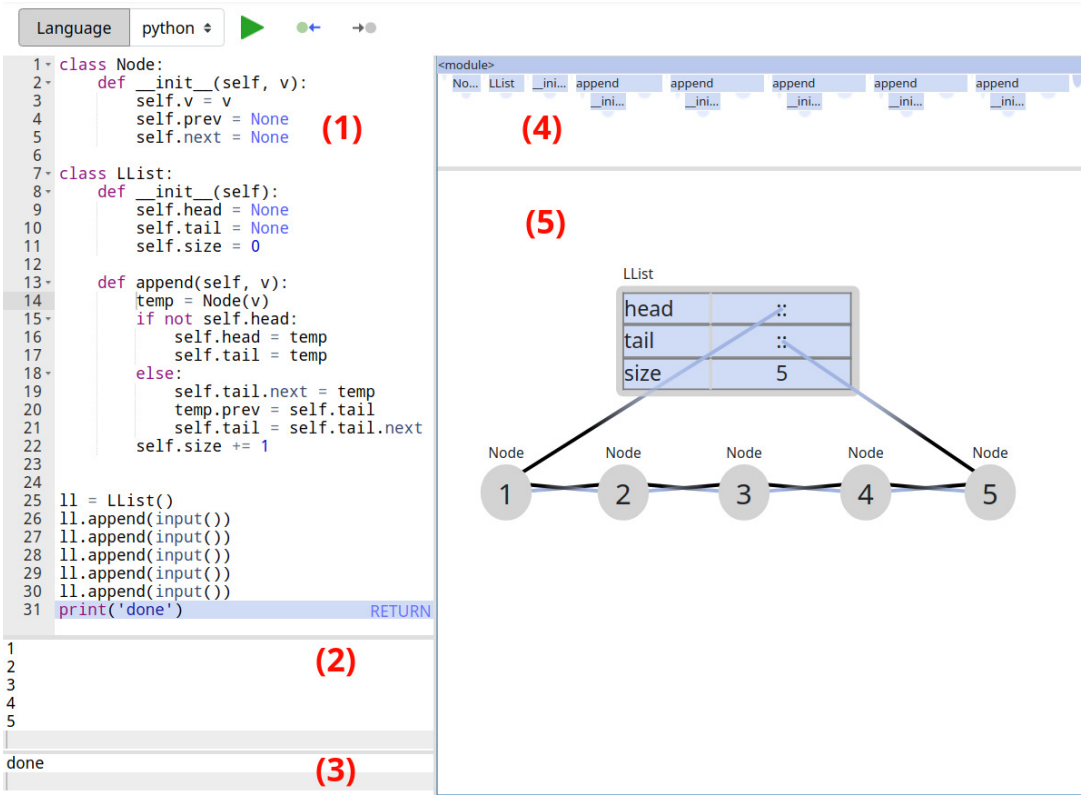

Figure 1: Willow is a web-based tool for Program Visualization Simulation where the user can modify the generated visualizations.

also DSA classes. We also intend to verify if our tool can help instructors in lessons preparations.

In the following sections, we discuss the background of our research (Section 2). We then describe our proposed tool (Section 3). We report a summary of our preliminary results with teaching assistants (Section 4). Finally, we discuss related works (Section 5) and conclude the paper (Section 6).

2 BACKGROUND

The main purpose of program visualization is to improve the understanding of how software works [15, 26]. It is done by representing abstract concepts of a program through its inspection. However, it is very challenging to find an effective way to map several aspects of a program and its programming language in visual elements in a way that they accurately represent the program operations [6].

Program visualization tools can be divided into two groups, Program Visualization Simulation (PVS) and Algorithm Visualization Simulation (AVS). PVS tools are focused on creating representations of the program state and its underlying structures (e.g., stack scopes and variables, objects in the heap, references, etc) [26] regardless the program source code and inputs. Although these tools show variable values and heap objects, they do not detect data structures in the program. Hence, they are not capable to create better abstractions to the notion of DSAs, leaving to the student to reason about it on a non-intuitive representation of the program memory.

AVS tools, on the other hand, try to cover abstract concepts of the DSAs through the inspection of the algorithm logic. This might help students to construct mental models and generalize their problem-solving patterns [26]. AVS tools are much less dynamic than PVS tools as they are specialized in providing visualizations to a small predefined set of DSAs, not allowing any modifications to the program source code.

There are already several tools for program visualization, and even more are created every year, as they continue receiving attention from educators and researchers. Most of these tools are tied to particular domains such as advanced debugging and profiling tools [4–6, 14, 21]. A survey with older tools focused on education can be found in [22] and [8]. Although some of these tools had good results among instructors and students, they are now outdated and not maintained anymore, making it hard to effectively use them.

One of the best known tools is Python Tutor[1] [10], this tool has a very large user base and it is the base for other tools such as OPT+Graph [6] and Omnicode [13]. These tools are categorized as PVS tools, as they were built to work with any simple program input, but they do not provide specialized visualization for specific algorithms, nor allow modifications to the generated visualization itself.

[1] http://www.pythontutor.com/

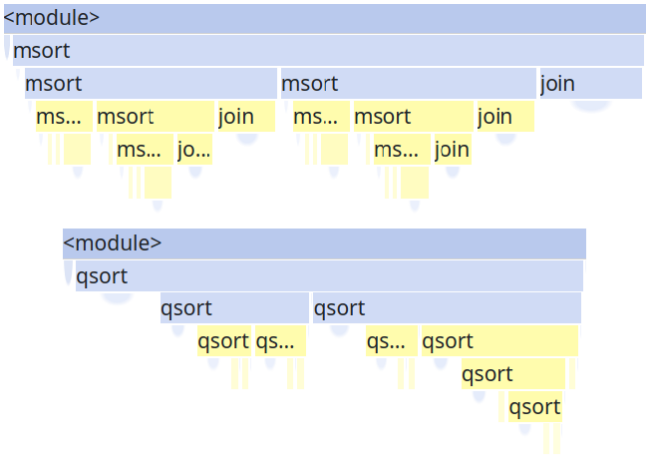

Figure 2: Stack story of recursive mergesort and quicksort algorithms respectively. Every element in the stack is clickable, by doing this, Willow will draw the beginning selected scope.

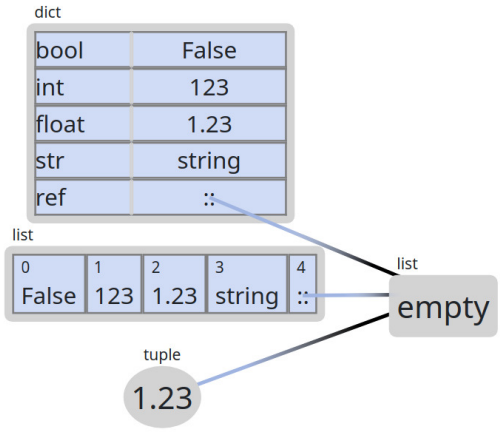

Figure 3: Willow visualization nodes. The node at the top is a map type, the nodes at the center are array type and the node at the bottom is a field node. Empty nodes may be objects or collections without elements or fields.

Other modern tools such as VisuAlgo[2], Algoviz[3] and OpenDSA[4] are also well known. These tools belong to the AVS category, they have well made visualizations, but for a limited number of DSAs. They also do not allow any modifications to the source code, making it impossible to modify the existing visualizations or create new ones.

3 WILLOW

We propose Willow, a web-based tool for generating interactive examples of DSAs. Willow is a PVS tool that allows the manipulation of its visual elements, allowing the creation of more expressive AVS-like visualizations. Because of that, Willow's primary targets are the instructors, as they use the tool to create visualizations for any program they want during class. Nevertheless, students can also benefit from directly using Willow, as well as potentially benefiting from the usage during class by the instructor.

3.1 Design

Figure 1 is a screenshot of Willow. The tool is divided into two main groups of components. The components on the left side are, respectively, (1) the code editor in which programs are written; (2) the input editor that is used to send any input data the program might need; and (3) the output editor that shows the data printed to the standard output and occasional error information. The right side contains the components with the role of showing program data visualization. The component on top (4) shows the history of a program's stack. Below (5), the current state of the program is displayed as a graph. The tool is meant to work under the following assumptions:

- The tool expects that all contents of a program must be in a single file and written in one of the supported languages. Since our main goal is to support introductory programming classes, we believe that requiring everything to be in a single file is reasonable. Language support is discussed in Section 3.3;
- Programs have limited size and restricted access to some libraries and features of the selected language, e. g., file system, threading support and network access. These restrictions were created to avoid any kind of abuse towards the tool usage;
- Program state is drawn as a graph of nodes and directed links, which represent objects and its references respectively. The way a node is drawn depends on the type of object and some user parameters that we discuss in Subsection 3.2

Program visualization tools can have an automatic or manual program navigation system. Tools that use automatic navigation will slowly pass through the program steps nonstop until the running algorithm reaches the end, whereas tools with manual navigation depend on the user input to take some action. The latter tools have different ways to implement user interactions to navigate through the program steps, such as clicking in buttons, using range sliders or moving the cursor to some point of the code.

We choose to implement a manual navigation strategy, combining two ways to interact with the program. It allows the user to go to an specific scope of the program just by clicking on any of the drawn scopes in the stack component showed in the Figures 2 and 1. By doing this, the first step on that scope is picked and drawn. The user can also click on the scope flaps as it allows fast navigation to intermediary or final points of the scope. The second way to navigate is using the keyboard arrows and modifier keys. This is used to advance and go back a single step at a time. By using the modifier keys, it can skip steps in function calls.

Algorithm implementations usually require some input data to be tested, Although this data might be provided by the program code itself, sending test data through an input stream is common, as the user does not need to change the program. Willow's input

[2]https://visualgo.net/
[3]http://www.algoviz.net/
[4]https://opendsa-server.cs.vt.edu/

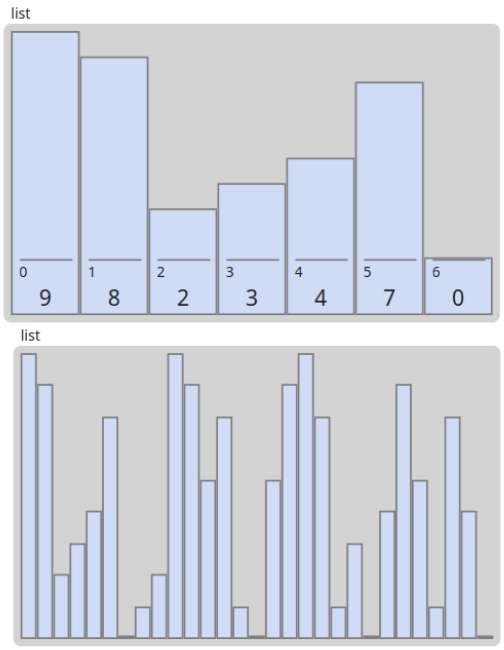

Figure 4: Numeric arrays shown as Bars nodes with different visualization parameters.

support is particularly useful because the program input can be used to generate different examples for the same algorithm.

3.2 Visualization

Willow's visualizations are generated from the program states and some parameters managed by the user. Each object of a program is by default associated to some kind of node. Nodes are the way Willow represents the objects in the graph. Willow has four types of nodes that are currently implemented, but new nodes can be added, allowing more algorithms to be better expressed. The nodes are shown in Figures 3 and 4 and they are:

- **array**: The array node shows fields of the underlying object as a horizontal list of its values. Array is the default node for most of the numerically indexable types, such as default implementations of arrays and lists. It is also the default for unordered but iterable types such as sets.
- **map**: This node displays object fields as a pair of columns of keys and values. This is the default for instances of user classes and map-like objects.
- **field**: Field is a node that can be used to show a single property of an array or object. It is mainly useful for representing user-created data structures, where objects may have many properties, making it potentially cleaner and easier for understanding.
- **bars**: This node shows numeric arrays as groups of bars. Each bar size is a proportion of the numeric value in the array index related to the biggest and smallest numbers in the same array. This node is further detailed in what follows.

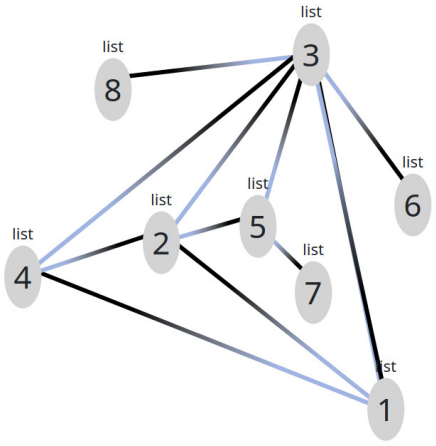

Figure 5: Graph made of python lists with references among them. These lists contain a number as one of their indices and are modified in the visualization to be shown as Field nodes.

Sorting algorithms are usually hard to be represented by tools specialized in data structures visualization. Since these tools tend to focus on user created objects and their composition in structures like linked lists or trees, they do not bother very much about drawing arrays used by such kinds of programs.

Willow has special support for default implementations of numeric arrays and lists, as these are data structures commonly used in sorting algorithms. These algorithms usually have many steps and if array objects are shown as a sequence of numbers, it may be tricky to perceive any differences at each step of the sorting. The implemented visualization is based in very common patterns found in many examples of sorting algorithms, where numbers of the array to be sorted are represented as vertical bars aligned side by side as shown in Figure 4. This arrangement helps the user to see what parts of the array are being sorted and how.

Willow allows object node types to be interchangeable if necessary, changing the way they are displayed. This allows map-like objects to be drawn as fields or array-like ones as maps, but not the opposite. Because of this and the ability to freely change nodes positions, generated program visualizations can be modified beyond a basic representation to express more accurately their abstract ideas.

Besides being able to switch among different kinds of nodes, each node has a set of parameters that change their properties and behaviours, influencing the way they are drawn. Figure 4 presents two arrays using different sets of properties, the array at bottom has its indices and values hidden and a smaller size per element. These parameters modifications can cause big modifications. Another example is shown in Figure 5. It is made only of list objects, which would be impossible to visualize in other PVS tools.

Another feature is that it detects groups of objects of common data structures like lists or trees. The collected group information is used to create automatic layouts for these structures and make them easier to understand. However, there are some conditions that need to be satisfied for a group of objects to be correctly detected as a data structure, and they are:

- The data structure must be made of objects of a single type. There are no restrictions about using only user defined classes, so language objects like arrays or dictionaries can be used to compose these structures;
- Objects of a data structure can contain references to other objects which store values, these values shall have a different type from the data structure objects type, otherwise they will be detected as part of the structure;
- For a data structure to be detected, it must be directly referenced by one of the variables in the program stack, or by an object that is referenced by a variable in the stack.

Automatic layouts of data structures are manually triggered by the user by double-clicking any node that belongs to a data structure, it applies the layout to the inner elements that compose the data structure but not the container. Figure 1 shows a doubly-linked list auto layout being applied.

3.3 Language Back-ends

Python has been chosen as the introductory programming language for CS1 courses across many universities. MIT and UC Berkley, some of the largest departments of Computer Science and many online courses such as Udacity [5] and Coursera [6] use this language [1, 10, 11]. Despite Python's growing popularity, many CS1 courses still use Java or even C/C++ as their introductory programming language.

Based on that, we implemented Willow's language support as plugable back-ends, allowing the use of Willow with many languages. Willow's multi-language support works through an intermediary program representation, which contains output, error messages, scopes, variables and objects in the heap of a program. Language back-ends are small programs that run and inspect received program inputs and source code written in a chosen language, generating the intermediary program representation and sending it back to Willow. We implemented back-ends for both Python and Java, as they are the most popular languages in CS1 courses.

4 EVALUATION

As mentioned before, we conducted a preliminary evaluation of the tool to validate its usability in CS1 classes. In our preliminary evaluation, we ran a small qualitative study. The 7 participants of our study were teaching assistants of CS1. The group of teaching assistants was composed of undergraduate students in different stages of the course, ranging from the first year to near the course conclusion. CS1 teaching assistants, besides being students that recently took such course, also usually give review lectures to the students. Therefore, they are a good target to start experimenting. The purpose of our study was to collect participants' perceptions about Willow and to identify future improvements. This study was performed with a previous version of our tool that only supported Python.

In the study, each participant received an introduction of how the tool works and videos showing the tool's features. The participants were free to try out the tool. Most of them tried simple operations with variables, functions, arrays and simple objects. Only two of the participants implemented linked lists. At the end of the study, the participants answered a questionnaire and a short non-structured debriefing interview session was conducted to collect impressions and opinions about the tool features.

From the participants' feedbacks[7], we can highlight some of them:

> I enjoyed the tool and would use it in monitoring, mainly to explain objects and references, which many students do not understand references.
>
> The tool is good, but not working with Java is a problem. For students, seeing code in another language can confuse and disrupt understanding.
>
> I just didn't like it because I can't go back in the visualization, I always have to run the program again and I have to navigate the whole program again to get to the point I was before.

In an overview of our preliminary results, we found that all participants had positive feelings about Willow. They all agreed in many statements such as the capabilities of the tool to demonstrate data structures and the ease of manipulating created visualizations. The study is, of course, very preliminary. However, it already provided useful feedback, such as implementing multi-language support and backwards navigation.

5 RELATED WORK

Willow is in the intersection between PVS and AVS tools. Some related PVS tools are UUhistle [25] and Jype [12], but these tools are old and it does not seem that they are active anymore. Nevertheless, Python Tutor and tools based on it, like Omnicode [13] and OPT+Graph [6], are more modern and more closely-related to Willow. Another tool that served as inspiration to Willow is Kanon [16]. Despite this tool not being focused on computer learning, it is very similar to Python Tutor and Willow.

Python Tutor and its derived tools are the main influences on Willow. They have common features such as similar objects visualizations, step-by-step navigation, input/output streams support and multiple language backends. However, Python Tutor falls behind Willow in the case of DSAs representation, as Python Tutor is a basic PVS tool. It does not show good visualizations for anything besides linked lists [10]. Moreover, it has an unmodifiable layout of nodes, so the user can not rearrange the nodes to help its understanding, which is bad for many DSAs visualizations.

Among PVS tools, Kanon is the only that allows moving nodes freely and has an automatic layout system, but it has too many constraints to work well. The generated representations are very fragmented and it creates arrows with references even for primitive types. This makes elements harder to understand, specially arrays, because they look like any other object. Willow implements a layout system which is inspired on Kanon but with fewer restrictions, making it easier to use.

On the AVS tools side, Algoviz and OpenDSA only provide visualizations for a very small amount of DSAs. VisuAlgo has more supported DSAs, with good visualizations for all examples. However, it does not allow modifications to the source code, and even simple examples like linked list reversing or many algorithms on graphs are missing.

[5] https://www.udacity.com/
[6] https://www.coursera.org/
[7] participants' feedbacks translated to English

		PVS		AVS	
	Willow	Python Tutor+	Kanon	Vizualgo	Algoviz
detailed algorithm visualizations	yes (after customization)	no	no	yes	yes
any program as input	yes	yes	yes	no	no
multi-language support	yes	yes	no	no	no
input/output support	yes	yes	no	no	no
automatic layout	yes (manually triggered)	no	yes	yes	yes

Table 1: Tools' features

6 CONCLUSION

This work proposes Willow, a tool to support Data Structure and Algorithms classes through the creation of interactive program visualizations that can be used by instructors and students in the teaching-learning process. Notable features are the ability to change visualization elements, allowing it to act more like Algorithm Visualization Simulation tools for any program the user inputs. Specialized nodes to show array data or select specific fields of an object allow the user to better control which and how objects are shown in order to make Data Structures and Algorithms easier to understand.

We ran an initial qualitative user study, showing that participants had positive impressions about Willow. However, as the tool is still in its early stages of development, it needs improvements to make the tool more flexible to visualize more programs.

Future work includes developing new ways to visualize objects; better automatic layout support for complex data structures like graphs; and general visualization improvements. We also intend to make an experiment with professors of Data Structures and Algorithms and, Introduction to Computer Science to collect better feedback, recommendations and make new findings about our tool.

ACKNOWLEDGEMENTS

We acknowledge support from FACEPE (IBPG-0751-1.03/18 and APQ-0570-1.03/14), and CNPq (409335/2016-9). This research was partially funded by INES 2.0, FACEPE grants PRONEX APQ-0388-1.03/14 and APQ-0399-1.03/17, and CNPq grant 465614/2014-0.

REFERENCES

[1] Muhammad Ateeq, Hina Habib, Adnan Umer, and Muzammil Ul Rehman. 2014. C++ or Python? Which One to Begin with: A Learner's Perspective. In *2014 International Conference on Teaching and Learning in Computing and Engineering*. IEEE, 64–69.

[2] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable programming: blocks and beyond. *arXiv preprint arXiv:1705.09413* (2017).

[3] Katrin Becker and Melissa Beacham. 2000. A tool for teaching advanced data structures to computer science students: an overview of the BDP system. In *Journal of Computing Sciences in Colleges*, Vol. 16. Consortium for Computing Sciences in Colleges, 65–71.

[4] Alexandre Bergel, Felipe Banados, Romain Robbes, and David Röthlisberger. 2012. Spy: A flexible code profiling framework. *Computer Languages, Systems & Structures* 38, 1 (2012), 16–28.

[5] ANM Imroz Choudhury and Paul Rosen. 2011. Abstract visualization of runtime memory behavior. In *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE, 1–8.

[6] Habibie Ed Dien and Yudistira Dwi Wardhana Asnar. 2018. OPT+ Graph: Detection of Graph Data Structure on Program Visualization Tool to Support Learning. In *2018 5th International Conference on Data and Software Engineering (ICoDSE)*. IEEE, 1–6.

[7] Stephen H Edwards, Daniel S Tilden, and Anthony Allevato. 2014. Pythy: improving the introductory python programming experience. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 641–646.

[8] Eric Fouh, Monika Akbar, and Clifford A Shaffer. 2012. The role of visualization in computer science education. *Computers in the Schools* 29, 1-2 (2012), 95–117.

[9] Denis Gračanin, Krešimir Matković, and Mohamed Eltoweissy. 2005. Software visualization. *Innovations in Systems and Software Engineering* 1, 2 (2005), 221–230.

[10] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 579–584.

[11] John Guttag. 2011. 6.00SC Introduction to Computer Science and Programming. https://ocw.mit.edu/

[12] Juha Helminen and Lauri Malmi. 2010. Jype-a program visualization and programming exercise tool for Python. In *Proceedings of the 5th international symposium on Software visualization*. ACM, 153–162.

[13] Hyeonsu Kang and Philip J Guo. 2017. Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 737–745.

[14] Ravi Khatwal and Manoj Kumar Jain. 2016. An Efficient Application Specific Memory Storage and ASIP Behavior Optimization in Embedded System. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS* 7, 7 (2016), 179–190.

[15] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education* 21, 1 (2011), 57–80.

[16] Akio Oka, Hidehiko Masuhara, and Tomoyuki Aotani. 2018. Live, synchronized, and mental map preserving visualization for data structure programming. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. ACM, 72–87.

[17] Michael C Orsega, Bradley T Vander Zanden, and Christopher H Skinner. 2012. Experiments with algorithm visualization tool development. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 559–564.

[18] Jibin Ou, Martin Vechev, and Otmar Hilliges. 2015. An interactive system for data structure development. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 3053–3062.

[19] Yizhou Qian, Susanne Hambrusch, Aman Yadav, Sarah Gretter, and Yue Li. 2019. TeachersâĂŹ Perceptions of Student Misconceptions in Introductory Programming. *Journal of Educational Computing Research* (2019), 0735633119845413.

[20] Yizhou Qian and James Lehman. 2017. StudentsâĂŹ misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1.

[21] Katarzyna Romanowska, Gurpreet Singh, M Ali Akber Dewan, and Fuhua Lin. 2018. Towards Developing an Effective Algorithm Visualization Tool for Online Learning. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2011–2016.

[22] Juha Sorva et al. 2012. *Visual program simulation in introductory programming education*. Aalto University.

[23] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)* 13, 4 (2013), 15.

[24] Juha Sorva, Jan Lönnberg, and Lauri Malmi. 2013. StudentsâĂŹ ways of experiencing visual program simulation. *Computer Science Education* 23, 3 (2013), 207–238.

[25] Juha Sorva and Teemu Sirkiä. 2010. UUhistle: a software tool for visual program simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. ACM, 49–54.

[26] Euripides Vrachnos and Athanassios Jimoyiannis. 2014. Design and evaluation of a web-based dynamic algorithm visualization environment for novices. *Procedia Computer Science* 27 (2014), 229–239.

[27] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. 2013. Facilitating code-writing in PI classes. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 585–590.