Comparative Analysis of Object Visualization Tools with Respect to Their Use in Education

Brandon Dennis and Ramyaa Ramyaa

***Abstract*— There are many visualization softwares for Object-Oriented Programming (OOP) in the literature. Each has different capabilities that allow them to visualize different aspects of OOP. However, to the author's knowledge, there is no survey of such visualization tools. Such a survey might help in comparing and contrasting the different tools across various dimensions. Further, a survey may uncover some crucial aspect of OOP that is not currently visualized, but could benefit from visualization.**

In this work, we survey visualization tools for OOP with respect to their use as an education tool. Specifically, we compare and contrast "JaguarCode", "jGRASP", "Javalina Code", "JIVE", "Python Tutor", "BlueJ", "Jeilot3", and "PandionJ". We consider the following dimensions of comparison: Detailed Graphics, Object Class Graphics, Single Instance View, Data Structure View, Additional Diagram Generation, and usability in introducing OOP concepts.

We found that while the existing technologies for object visualization are sufficient for most applications in education, there is a lack of tools that target introducing OOP. In particular, the tools lack features designed specifically to educate students on fundamental ideas of OOP such as object design.

***Index Terms*— computer science education, object-oriented programming, software, visualization**

I. Introduction

Object Oriented Programming (OOP) can be a difficult concept to master. Learning what an object is and how to design object classes can be particularly difficult. Visualization tools can help with this. A graphical representation of objects and what they encapsulate can help them understand how objects can be built and used.

There are many visualization softwares for Object-Oriented Programming (OOP) in the literature. Students may be confused at many different levels and so, different visualization tools may choose to focus on different levels of detail. Graphics such as Class Diagrams can be good for aiding in object design, whereas more complex visuals like sequence diagrams can assist in a student's understanding of the order of events occurring in a program. Visualizing individual objects can be helpful for understanding how data in an object can change over time while a tool allows for the visualization of a complex data structure may be more desirable for a user who needs to see how objects interact with each other.

However, to the author's knowledge, there is no survey of such visualization tools in an educational context. Such a survey comparing and contrasting existing tools could highlight a gap in current visualization software. Here, we provide such a survey focusing on using visualization tools in OOP education. Specifically, we compare and contrast "Jaguar Code", "jGRASP", "Javalina Code", "JIVE", "Python Tutor", "BlueJ", "Jeilot3", and "PandionJ". We found that object visualization is most commonly done in a format similar to the UML Class Diagrams, but there are several approaches used by different modeling tools. Some give a surface level view of objects, while others can be used to model large-scale data structures. Though not all of these tools were specifically designed for education, many of the authors of these tools had intended education to be one of the tool's uses, and had reported experimental results of the tools' use in an educational setting. We consider the following dimensions of comparison: Detailed Graphics, Object Class Graphics, Single Instance View, Data, Structure, View, Additional, Diagram, Generation and usability in introducing OOP concepts.

We conclude that, while the tools had varying degrees of performance across the dimensions of our analysis, they all miss concepts that could make them more effective as educational tools. For instance, they all lacked a focus on foundational principles of OOP such as object design and the difficulty students have transitioning from procedural-style languages to an Object-Oriented approach. Instead, these tools were intended for users who already have a fundamental knowledge of objects and how to design them. Adding more features designed to educate students on fundamental ideas of OOP like object design would be a step in this direction

Received November 18, 2021, Accepted December 22, 2021, Publish online January 12, 2022.

Both authors are affiliated with the New Mexico Institute of Mining and Technology, Socorro, NM 87801 USA (e-mail: brandon.dennis@student.nmt.edu and ramyaa.ramyaa@nmt.edu).

II. Overview of Existing Technology

While all of the tools covered in this analysis are used to visualize object-oriented code in some form, the level of detail and capabilities of each tool are not the same across the board. Some tools are simplistic in their graphics and only aim to give the user a base-level view of data, while others allow the user to view full scale data structures in detail. This section gives a brief introduction of each tool.

A. JaguarCode

JaguarCode ([1]) is a web-based programming tool designed for dynamically visualizing Java code. JaguarCode generates static diagrams for classes, as well as dynamic views of each instance of an object used in a program being ran in the tool. Additionally, the tool visualizes relationships between associated objects such as inheritance or implementation as shown in Fig. 1.

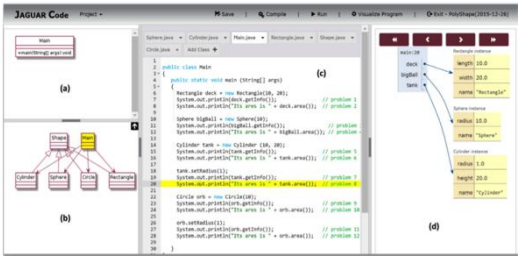

Fig. 1. An annotated screenshot of JaguarCode taken from [1]. The left-hand side depicts the generated Class Diagrams, the middle section is for user-given code, and the right-hand side is used to visualize running code.

B. *jGRASP IDE*

The jGRASP IDE ([2]) is a Java programming tool with a wide range of visualization capabilities. It is mainly focused on visualizing multi-object data structures (such as linked lists shown in Fig. 2.) rather than individual classes, but there are some UML visualization capabilities.

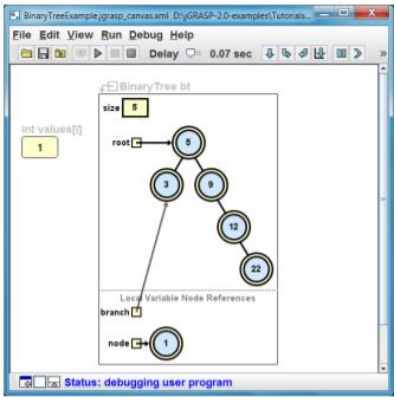

Fig. 2. A view of a binary tree generated by the jGRASP IDE taken from [2].

C. *JavalinaCode*

JavalinaCode ([3]) is an online Java visualization tool. It uses a cloud service to allow users to create Java projects, compile code, and visualize programs. The tool generates Class Diagrams for each class in the project and table-like graphics for each object used in a running program much like the JaguarCode example in Fig. 1.

D. *JIVE*

The Java Interactive Visualization Environment (or JIVE) [4] is a plugin for the Eclipse IDE designed to dynamically visualize Java code. Unlike some of the other tools presented so far, JIVE does not offer Class Diagrams as a form of static visualization. Instead, the tool generates Contour Diagrams and dynamic, table-like graphics for objects used in the running program (shown in Fig. 3.). JIVE also offers Sequence Diagram generation.

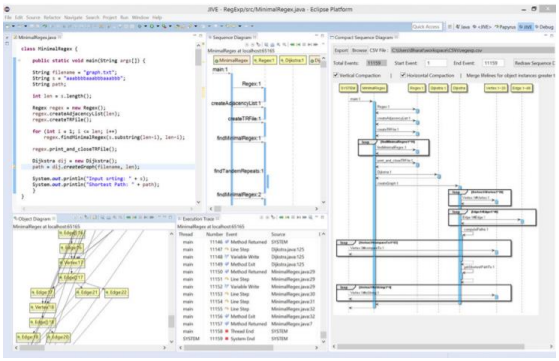

Fig. 3. Screenshot of the JIVE user interface taken from [4]

E. *Online Python Tutor*

Online Python Tutor ([5]) is a web-based programming tool intended for both students and instructors that allow the user to step through Python code with the aid of visualizations. The tool allows users to step both backwards and forwards step-by-step through a program while providing a graphic representation of the execution stack, global variables, and local variables. Complex objects are given their own graphic representations depicting the current value of their attributes such as the graphics in Fig. 4.

Fig. 4. Graphics generated by the Online Python Tutor taken from [5].

F. *BlueJ*

BlueJ ([6]) is a simplistic Java visualization tool that allows users to interact with objects graphically. The visual representation of objects generated by BlueJ are lacking in detail and offer no description of a class's methods or attributes, which can be seen in Fig. 5. They are, however, capable of visualizing inheritance relationships and individual instances of an object.

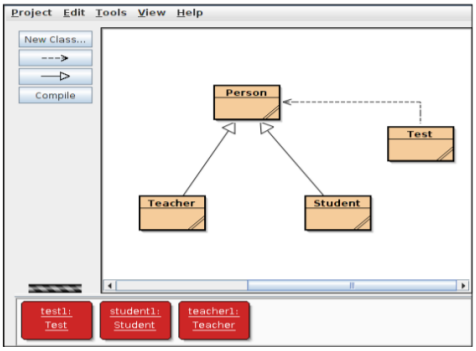

Fig. 5. Visualization generated by the BlueJ tool taken from [6].

G. *Jeliot 3*

Jeliot 3 ([7]) is an evolution of the Jeliot visualization tool for Java. This tool allows the user to view and interact with the graphics it generates alongside the written code. Similar to JIVE, Jeliot 3 focuses more on visualizing programs as a whole rather than individual classes. The graphics generated by the tool include graphics similar to a contour diagram with separate graphics for object variables (see Fig. 6. for details).

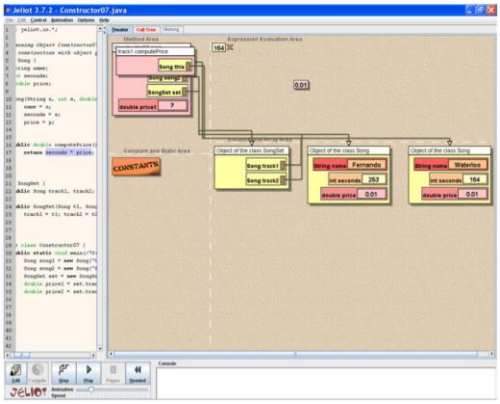

Fig. 6. Screenshot of the Jeliot 3 visualization tool taken from [7].

H. *PandionJ*

PandionJ ([8]) is an Eclipse plugin designed to visualize Java code. The plugin allows users to see objects and function stacks visualized alongside the running code. PandionJ also allows for step-by-step code execution and debugging.

JaguarCode, jGRASP, JavalinaCode, Online Python Tutor, BlueJ, Jeliot 3, and PandionJ were tested in educational settings and the results were detailed in [1] [2] [3] [5] [6] [7] [8] respectively.

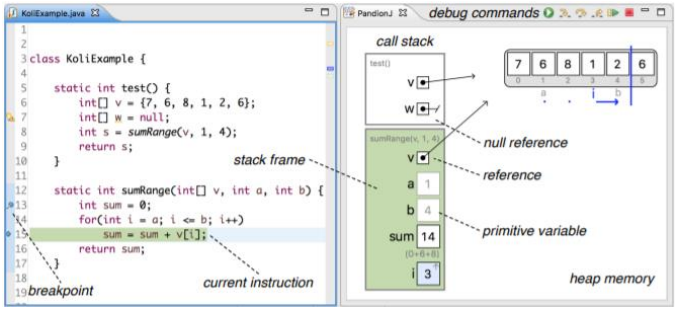

Fig. 7. Screenshot of the PandionJ plugin taken from [8].

III. Tool Comparison

We first give brief overview of the comparison criteria. The functionalities of these tools being compared in this paper can be grouped into 6 categories: visualization detail, object class graphics, individual instance view, data structure view, additional diagram generation, and usability in introducing OOP. A reference of what criteria a given tool offers can be found in Table 1.

A. Visualization Detail

The amount of detail provided by a visualization tool can greatly impact usability. Some of the tools covered in the previous section give graphics that allow the user to view the names of individual instances generated by the program, or even the data types of an object's attributes. This information can greatly benefit the user, especially when using multiple objects of the same class in a single program.

B. Object Class Graphics

A key feature of these visualization tools is the ability to produce graphics depicting the objects being used in a program and the data being stored inside them. Some tools, however, offer a broader view of the data as well in the form of Class Diagrams (or their own equivalent form). These diagrams allow the user to view the characteristics of an entire class of objects rather than just the data in an object instance. This information can be helpful when designing objects since diagrams like these can often serve to condense large amounts of code into a more digestible format.

C. Individual Instance View

The ability of a tool to generate a visualization of individual instances being used in a program is arguably what classifies a program as a visualization tool. Each tool being reviewed is capable of this functionality, although with varying levels of detail provided by each.

D. Data Structure View

On top of being able to visualize individual objects, some visualization software allows users to generate graphical representations of complex data structures like binary trees or linked lists. This can be an especially useful tool when creating programs that utilize data structures like these since visuals like this can help the user determine if each structure is functioning correctly in the large scale.

E. Additional Diagrams Generated

While being able to generate graphics for a single snapshot of a program's runtime is certainly useful, sometimes additional data is needed or desired. This is where visual aids such as Sequence Diagrams can be useful. A Sequence Diagram is a single graphic that can describe the interactions between objects over time. Typically, these are generated manually, but some visualization tools give the user the ability to generate these views along with the traditional graphics.

F. Usability in introducing OOP

In order for a tool to be useful in introducing OOP to students, it must offer details about the fundamental elements of OOP. For example, teaching students about proper object design is a vital part of OOP education. Another helpful feature would be a method of aiding a student's transition from procedural programming to an Object-Oriented approach. Features like these would provide a helpful teaching aid to students just beginning their programming education.

TABLE 1
Tool Comparison Overview

	Jaguar Code	jGRASP	Javalina Code	JIVE	Python Tutor	BlueJ	Jeliot 3	PandionJ
Detailed Graphics	X	X	X	X	X	X	X	X
Object Class Graphics	X		X				X	
Single Instance View	X	X	X	X	X	X	X	X
Data Structure View		X						X
Additional Diagram Generation	X			X				
Usability in introducing OOP								

This table serves as a brief reference of what comparison criteria are and are not offered by each tool covered in this paper.

IV. Tool Analysis

A. JaguarCode

JaguarCode's dynamic visualizations are displayed in the right-hand pane of the window and include a visual representation of the function stack, as well as names of the variables contained in each frame. If necessary, JaguarCode also generates a graphic for objects in each frame with an arrow back to the frame they are used in as displayed in Fig. 1. While JaguarCode does not have the ability to visualize large scale data structures, it can generate sequence diagrams for user given code.

JaguarCode is a good visualization tool for students who already have some programming knowledge but struggle with understanding objects as a concept. The tool offers some debugging tools along with dynamic visualization for each stage of the code, which can be very useful in understanding how objects can change over the course of a program's execution. The static Class Diagrams also help the user understand the content of a class file, as well as gains some knowledge of how an object can be used. The addition of a sequence diagram generation feature can also be useful for more advanced users who plan on using JaguarCode for more sophisticated projects.

B. jGRASP IDE

The jGRASP IDE is a great tool for visualizing multi-object data structures. It allows users to visualize commonly used data structures such as linked lists, hash tables, and binary trees. jGRASP also common debugging features like user given breakpoints and the ability to step through code. What jGRASP offers in data structure visualization, however, it lacks in general object visualization. There are no static visualizations for object types, though the names and values of instance attributes are given in the generated graphics.

While it may not be a particularly useful tool for learning how to design classes, the jGRASP IDE is a great tool for any application that requires tracking data in a large-scale data structure. It may also be useful for students who are learning about these types of structures and could use a visual aid for comprehension.

C. JavalinaCode

JavalinaCode is fundamentally similar to both JaguarCode and the Online Python Tutor in the sense that all of them visualize the

function stack and point to individual instances of objects. In further similarity to JaguarCode, JavalinaCode also gives static visualizations for object classes along with visual representations of inheritance and other object relationships. Unlike JaguarCode, however, JavalinaCode lacks the ability to generate sequence diagrams over the course of program execution.

D. JIVE

The JIVE visualization tool may be the most detailed visualization tool covered in this paper in terms of the level of detail given for an individual object. JIVE offers users a table view of each object where attribute names, values, and data types are given. The fact that this information is given is especially important given that JIVE does not give users a static Class Diagram view of objects. Instead, the tool focuses on giving users as much information about an object as possible directly in the dynamic view. JIVE also gives users the option of generating Sequence Diagrams, which can also be useful for understanding interactions between objects.

E. Online Python Tutor

The Online Python Tutor acts as a basis for tools like JaguarCode and JavalinaCode. While Online Python Tutor may not offer as many features these tools, it was published well before either of them and likely served as a basis for both given how similar their visualization styles are to Online Python Tutor. And, while this tool may not be as feature heavy as others, it does still offer a fairly streamlined and convenient view of data. That, along with the convenience of being an web-based tool and the ability to execute a program step-by-step make the Online Python Tutor a good education tool.

F. BlueJ

BlueJ is a bare bones approach to object visualization. Its figures offer little detail (seen in Fig. 5.), and no general object class graphics are given. BlueJ does, however, name object instances and represent inheritance graphically, which is useful information to have visualized. As the oldest tool covered by this paper though, BlueJ is a solid foundation for the field of object visualization to build on.

G. Jeliot 3

Jeliot 3 is another tool that allows a user to step through execution while giving dynamic visualizations of variables and objects in their code. While no static diagrams are generated by the tool, Jeliot 3 does give a description of data types in its graphics. Another interesting feature of Jeliot 3 is the ability for users to interact with the visualizations it generates. For example, Fig. 6. shows a user moving objects around in the data view.

H. PandionJ

PandionJ offers a similar experience to that of the Online Python Tutor with a few added features. Like the Online Python Tutor, PandionJ offers detailed visualizations of single object instances, but does not offer general class descriptions. PandionJ does also give the user visual indication of null references (a node cut off by a slash) and differentiates between objects and primitive data types (primitive data type attributes have desaturated variable value boxes).

V. Discussion and Conclusion

Visualization software fills a gap in OOP education left by traditional instruction alone. These tools allow students to experiment in a safe environment at their own pace, and visualization allows students who are less familiar with code to gain a better understanding of their programs. However, there are a few areas where current visualization tools could improve.

The tools do not focus on introducing OOP concepts. To be effective at this level in OOP education, a tool must have functionalities that address more fundamental principles of OOP. Many of the tools covered in this article, for example, seem to be designed for users who already know how to design objects. A foundation in OOP is almost a pre-requisite for using these kinds of tools, which doesn't help students just learning how to program in an Object-Oriented space. A potential solution to this issue is to offer a series of guided tutorials that teaches the user some of the more fundamental concepts of OOP like object design, the difference between private and public attributes or methods, etc. Giving a student these sorts of instructional guides before allowing them to interact in the tool freely may help students build a strong foundation in OOP.

Another area that current visualization software is lacking is the level of detail offered by their graphics. While a majority of the tools covered in this paper give some way of identifying object instances, many of them do not give this information in a convenient location. For example, both JaguarCode and the Online Python Tutor name object instances in their visualization of the stack but not in the graphics generated for the objects themselves. Another detail many of these tools do not offer in their graphics is the data type of an object's attributes. The tools that offer static Class Diagram generation have data types listed, but this level of detail is dropped in the dynamic visualization. Offering instance names and attribute data types all in one place provides the user with a greater level of detail in a more convenient location. This improvement can help with usability, especially for a novice in OOP.

The last area we believe visualization software can improve is by giving users who have programming experience but are new to OOP a way to transition between the two styles of coding. The transition between procedural programming and OOP can be jarring, and many students struggle with adapting to the idea of objects that can contain their own data and methods. While some procedural languages have mechanisms that can be considered similar to objects (such as structs in C), shifting your thinking to fit with a near-fully Object-Oriented environment like Java can be difficult. Visualization software currently doesn't do anything to aid in this transition. A possible way to accomplish this is to generate a C-like pseudocode representation of user given code along with more traditional visualizations.

VI. References

[1] J. Yang, Y. Lee and K. H. Chang, "Evaluations of JaguarCode: A web-based object-oriented programming," *Journal of Systems and Software*, no. 145, pp. 147-163, 2018.

[2] J. H. Cross II, T. D. Hendrix, L. A. Barowski and D. A. Umphress, "Dynamic program visualizations: an experience report," in *Proceedings of the 45th ACM technical symposium on Computer science education*, Atlanta, 2014.

[3] J.-s. Yang, *JavelinaCode: A Web-Based Object-Oriented Programming Environment with Static and Dynamic Visualization*, Auburn, Alabama, 2016.

[4] S. Jayaraman, B. Jayaraman and L. Demian, "Compact visualization of Java program execution," *Software: Practice and Experience*, vol. 47, no. 2, pp. 163-191, 2017.

[5] P. Guo, "Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia," in *The 34th Annual ACM Symposium on User Interface Software and Technology*, 2021.

[6] B. Y. Alkazemi and G. M. Grami, "Utilizing BlueJ to teach polymorphism in an advanced object-oriented programming course," *Journal of Information Technology Education*, vol. 11, pp. 271-281, 2012.

[7] M. Ben-Ari, R. Bednarik, R. B.-B. Levy, G. Ebel, A. Moreno, N. Myller and E. Sutinen, "A decade of research and development on program animation: The Jeliot experience," *Journal of Visual Languages & Computing*, vol. 22, no. 5, pp. 375-384, 2011.

[8] A. L. Santos, "Enhancing Visualizations in Pedagogical Debuggers by Leveraging on Code Analysis," in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, Koli, 2018.

Brandon S. Dennis was born in 2000 in Socorro, New Mexico, USA. He is an undergraduate student at the New Mexico Institute of Mining and Technology in Socorro, New Mexico and is currently pursuing a Bachelors of Computer Science with an accelerated Masters program.
His current field of interest is computer visualization for use in education.

Ramyaa Ramyaa is an assistant professor of Computer Science at New Mexico Tech. She did her PhD at Indiana University, USA, Post-doctoral research at Ludwig Maximilian University, Munich and has masters degrees from Carnegie Mellon University, USA and University of Georgia, USA. She was a fellow at Simons institute of Theory, UC Berkeley, USA.
Her primary fields of research are Theory of Computation (and complexity) and Logic, focusing on implicit complexity (relating logical complexity of concepts to computational, resource-based complexity). Her secondary research area is Artificial Intelligence, focusing on machine learning.