

Initial Evaluation of JaguarCode: A Web-Based Object-Oriented Programming Environment with Static and Dynamic Visualization

Jeong Yang

*Dept. of Computing and Cyber Security
Texas A&M University-San Antonio
San Antonio, U.S.A.
jeong.yang@tamusa.edu*

Young Lee

*Dept. of Electrical Engineering and
Computer Science
Texas A&M University-Kingsville
Kingsville, U.S.A.
young.lee@tamuk.edu*

Kai H. Chang

*Dept. of Computer Science and
Software Engineering
Auburn University
Auburn, U.S.A.
changka@auburn.edu*

Abstract—Various visualization techniques have been adopted to educational Object-Oriented Programming (OOP) environments. Some provide software development with visual notations without source code, while others support programming with visual aids. Our research supports Java programming along with static UML class diagrams and dynamic execution trace of program synchronized in a web-based programming environment - JaguarCode. It aims to help students better understand static structure and dynamic behavior of Java programs, and object-oriented design concepts. This paper reports on an initial evaluation of JaguarCode to investigate its effectiveness and user satisfaction through quantitative and qualitative experiments. The experimental results revealed that having both static and dynamic visualizations did positively impact the correctness of program understanding and tracing problems, and the visual representations did affect students' understanding on program execution of the problems to higher accuracy. It was also observed that students were satisfied with the aspects of those visualizations provided in JaguarCode.

Keywords—*object-oriented programming, static and dynamic visualization, web-based programming environment, Java*

I. INTRODUCTION

Object-Oriented Programming (OOP) is one of the core areas in Computer Science, and learning OOP becomes a major challenge in Computer Science education. While Java programming language has been widely used in teaching and learning of an OOP, studies have identified that it is difficult for students to learn due to the underlying OO concepts and principles, such as encapsulation, abstraction, inheritance, and polymorphism [1, 2]. With inheritance, polymorphism, and dynamic binding features, objects in OOP interact with each other asynchronously, and object flows are difficult to track. Learning difficulties could be also originated from different sources, such as complexity and domain of a problem, program design, programming environment, programmer's logical thinking ability and programming skills as well as OO concepts and principles. Comprehension of structure and behavior of an OO program is a crucial component of the process of programming learning.

A student's development environment is another factor that influences learning OOP. It requires students to manage issues, such as platform dependencies and conceptual understanding of classes, objects, and Object-Oriented Design (OOD). Visual techniques in various formats have been applied to educational programming environments. Some provide visual notations without source code, while others support a single aspect, structural or functional behavior, of the program. Studies also have found that novice students often encounter difficulties in installing Integrated Development Environment (IDE) and plugins, and setting up and modifying system environment variables on their own machines [3, 4, 5]. In addition, IDEs, like Dev-C++ and Visual Studio, run on Windows Operating System (OS), while Mac OS is in high demand these days.

To enhance OO program comprehension, to improve OOD concepts, and to deal with the platform dependency issues, our research uses an approach that integrates structural and behavioral aspects of OOP in a web-based programming environment, JaguarCode (formerly JavelinaCode [7, 8, 9, 11]). JaguarCode provides synchronized static and dynamic representations of visualizations along with source code, the static structure of a Java program and the dynamic runtime state of program execution. When a student is writing a line of code, its corresponding structural information of the program is highlighted in UML class diagrams, and the run-time state of program execution is synchronized with the code. Through these synchronized multi-view visualizations in a single window, both structural and functional feedback of the current line of the source code is immediately provided to the student.

The major motivation for this work is the lack of an effective platform-independent OOP environment integrated with both static and dynamic visualizations. When inheritance, polymorphism, and dynamic binding are taught using class diagrams and sequence diagrams with source code, students must simulate the execution of programs in their minds to understand how they work. As the combined visualization of static and dynamic aspects was suggested [6], visualizing both aspects corresponding to the source code is anticipated to help students better understand the OO programming and design concepts.

II. RELATED WORK

Some of the educational programming environmental tools are reviewed in this section, based upon how the OO features are highlighted in visualization in terms of static and dynamic components of an object-oriented program. It provides a brief description of tool's purpose and interface, which visualization techniques are used, and how well they are integrated with source code. While some provide only static visualization, dynamic runtime execution with object, or both, none of these tools support both with an integrated user interface. The comparative analysis of these tools was conducted to measure the time required to download and install them [7, 9]. Unlike JaguarCode, they must be downloaded and installed as a stand-alone program or plugged into Eclipse or NetBeans.

A. BlueJ

BlueJ is an IDE that allows novice students to interact with objects and link source code with its UML class diagram. When a Java project is created/opened, a main window displays a UML class diagram representing its static structure, so students can interact with classes in the project. It was identified that BlueJ's capability of linking source code and visualizations helped students learning in the cognitive domain [11]. The studies also showed that BlueJ was useful for students in the first-year OOP course and it helped them learn object-oriented paradigm [10, 11]. BlueJ's (Version 3.1.7, released in February 2016) user interface was found to be straightforward to build a new project, but running a project was not integrated with the other features [12]. The program source code, the output of the programs, and the display of UML class diagrams are displayed in a separate window [9], [12]. This makes linking them together more difficult to understand and follow. BlueJ relies on the static class diagrams that are interactive for learning, but no run time visualization technique employed, thus it seems difficult to detect run time errors of source code [12].

B. Jeliot 3

Jeliot 3 is a program visualization tool designed for novice students to learn both procedural and OOP with a supporting animation of the data flow, control flow, and expression evaluation [13, 14]. Jeliot 3 (Version 3.7.2, released in March 2014) provides a visualization of a large subset of Java programs, supporting object-oriented concepts. The main structure of the animation frame used in visualization includes method, expression evaluation, constant, and instances frames. UML class diagram-like notation is used to visualize objects shown as boxes containing attributes and values and references shown as lines connecting the object and its variable. The link between program source code and its corresponding visualization area is also synchronized and highlighted to identify them together at a time. But the visualization techniques are restricted to the theater display area, offering a dynamic behavior of its running state with the lack of meaningful static visualization.

While studies found that students could benefit from Jeliot 3 animations to debug programs and the animation made it easier to discuss programming concepts between students and instructors [13], it was also found that novice student

programmers had difficulty to understand the animations [14]. Jeliot 3 does not support adding multiple files into a single project, limiting students' understanding of how classes are distinct from one another [9].

C. jGRASP

jGRASP is an IDE for program visualizations that supports structure identifier viewer to analyze source code and detect arbitrary data structures. It allows interactions with its dynamic viewers to support understanding of data structures [15, 17]. jGRASP (Version 2.0.2_01, released in March 2016) allows line by line program execution through the use of debugger and canvas window. Execution of a Java project in the canvas can be paused and elements from the debugger or work bench can be dragged into the canvas window. By dragging each of the created objects into the canvas, users can observe the behavior of the object. Users also can generate a simplified version of a UML class diagram of the project.

For the code understanding experiments using jGRASP data structure viewers, it was found that students could detect and correct logical errors more accurately using the viewers than the traditional methods of visual debugging for singly linked lists [16]. Using data structure visualization on certain programming tasks, other experiments also have shown that students in the experimental group consistently performed better than a control group [15]. The results of the experiments revealed positive effect of the canvas viewers that students used the viewers did code faster and had fewer errors in program development, and found more errors and faster in existing code than students not using the viewers [17]. However, the source code, the canvas window, and a class diagram are displayed in multiple windows. To gain a full understanding of the program, students need to transit among them, which make it difficult and disconnected by the use of multiple windows [9].

D. AguiaJ

AguiaJ (Version 1.1, released in October 2013) is a pedagogical tool and Eclipse plug-in that uses a visual representation of metaphors to aid understanding of OO concepts [18]. These metaphors support user interaction with objects, polymorphic behaviors by associating reference types and structural elements of objects, and inheritance concept by visualizing inherited objects and identifying the difference between the objects of super and sub classes. AguiaJ window comprises two major areas: a class area and an object area. Defined classes in source code are adapted into the class area, and created objects are populated in the object area.

An evaluation study has concluded that AguiaJ was usable for interactive lecturing and exercising, and teaching OOP with the domain of image manipulation [19]. It was also discovered that mapping source code and object illustration is hard due to the multiple visualization windows displayed [12]. While AguiaJ supports flexible static visualization of source code, classes and objects are not bound to nor dependent on a particular runtime environment of the program, making it not effective to check runtime errors [9]. When inherited methods cannot be used as inputs to a particular object, problems become not-traceable for checking runtime behavior [9].

E. JIVE

JIVE is an interactive program execution environment, supporting the visualizations of the runtime state and the call history of a Java program. Its approach is to facilitate program understanding and the comprehension of runtime execution of object-oriented programs in Java, through displaying runtime object structures, providing object states in multiple views, visualizing the history of program execution with sequence diagrams, supporting forward and backward program execution, and producing clear drawings of the object structure and method-invocation sequence [20, 21]. The JIVE interface itself includes two UML diagrams, an object and sequence diagram, and a contour model. A complete run-time state of a program is visualized through a contour (object) diagram, showing an object structure with method activations in object contexts. A history of program execution is visualized, using a time sequence diagram.

The experiences of using the extended notation of contour model found that the technique of using contour diagrams were appropriate for object-oriented programs, and the diagrams were useful for debugging, particularly to make a difference between a user's imagined structure and an actual structure created [22]. Although the time sequence diagrams generated in JIVE have proven effective in explaining the behavior of design patterns and program structures [21], it made the history of the diagrams very complicated [21]. To use JIVE in Eclipse, users must properly configure the debugger and to enable the debugger for a project, users need to create a launch configuration for the project, edit the configuration to specify that JIVE must be used for debugging, and open the JIVE window perspective for checking the visualization of the program execution. The project must be run at least once before configurations can be modified to include JIVE [9].

III. JAGUARCODE

JaguarCode uses an approach to integrate structural and behavioral aspects of OOP in a platform independent web-based environment. It provides synchronized static and dynamic visualization of Java programs at line level and a full overview of a project under development. A student can access JaguarCode through a front end web browser (<http://www.jaguarcode.com>). An overview of the system and applied design principles are presented in [8, 12]. The system is also designed to complement and reinforce an easier and flexible file management capability not only to save and open project files to and from the cloud server but also do the same on a local computer.

A. System Implementation

JaguarCode is being implemented with a front end, written in HTML5, CSS3, and jQuery, and a back end, written in PHP and Java. Ace, an embedded open-source code editor, is fully integrated into the environment [25]. Ace editor comes with numerous advantages such as syntax highlighting, automatic indentation, facility to handle large files, clock highlighting, and dragging and dropping source code from other resources. To generate UML class diagrams for the static information of Java source code, Plant UML, an open-source tool that

converts textual code description to draw UML diagrams, is integrated into the system [26]. For the run-time state visualization of program execution, Java Visualizer [27] is customized and integrated [7]. JaguarCode has been developed on the Amazon Web Services (AWS) cloud computing platform on the back end virtual server, running Ubuntu 14.04 operating system, Apache 2 HTTP server, and MySQL database server with php5 and Java 8. PhpMyAdmin is used to handle the administration of the MySQL and to interact with its databases for managing users and project files.

B. User Interface

The user interface of JaguarCode is presented in Fig. 1. The interface comprises four main components: static UML diagram areas (a) and (b), an editor area (c), and a dynamic run-time state visualization area (d). The editor area displays the active Java code a student user is working on, and by selecting a tab (Add Class), the user can create multiple Java files and add them into a project. When a new class is added, the default code, representing the basic structure of a Java class, is presented to the user to start immediately changing the existing code. After 'Visualize Program Execution' button is clicked, for each line of the code, its corresponding class is highlighted in the compact class diagram in (b) and dynamic information of data is synchronized in the run-time state visualization in (d). Using a set of buttons, each line of the code can be traced and analyzed by stepping forwards and backwards.

Three sets of UML diagrams are generated: (a) one for the active Java program in the editor, (b) one compact diagram for the whole project, and one detailed diagram containing all the information related to the current project. When an enlarged icon  in area (b) is clicked, the detailed UML class diagram for the project is illustrated in a separate window (see Fig. 2). The detailed diagram in the new window shows all classes created in the project including all relationships among classes, such as association, inheritance, and interface. This will become helpful, especially, when the program gets larger with more classes added to the project.

C. Static and Dynamic Visualization

Plant UML and Java Visualizer are customized and integrated into the JaguarCode interface. A static aspect of the source code is visualized, using a customized Plant UML class diagram, and a dynamic aspect of the program execution is visualized, using a customized Java Visualizer. In both cases, program execution happens in memory. All Java files made from the editor for a project are merged into a single class file, which will serve as a main class. Other classes in the project will be inner classes for the main class.

Plant UML is a web-based open-source visualization tool that allows users to create UML diagrams from a plain textual description [26]. To draw a UML class diagram and to integrate it into JaguarCode, a textual input from the merged single Java file is produced with four different regular expressions defined [7], which are used to detect classes, extended classes, implemented interfaces, dependencies, methods, and variables.

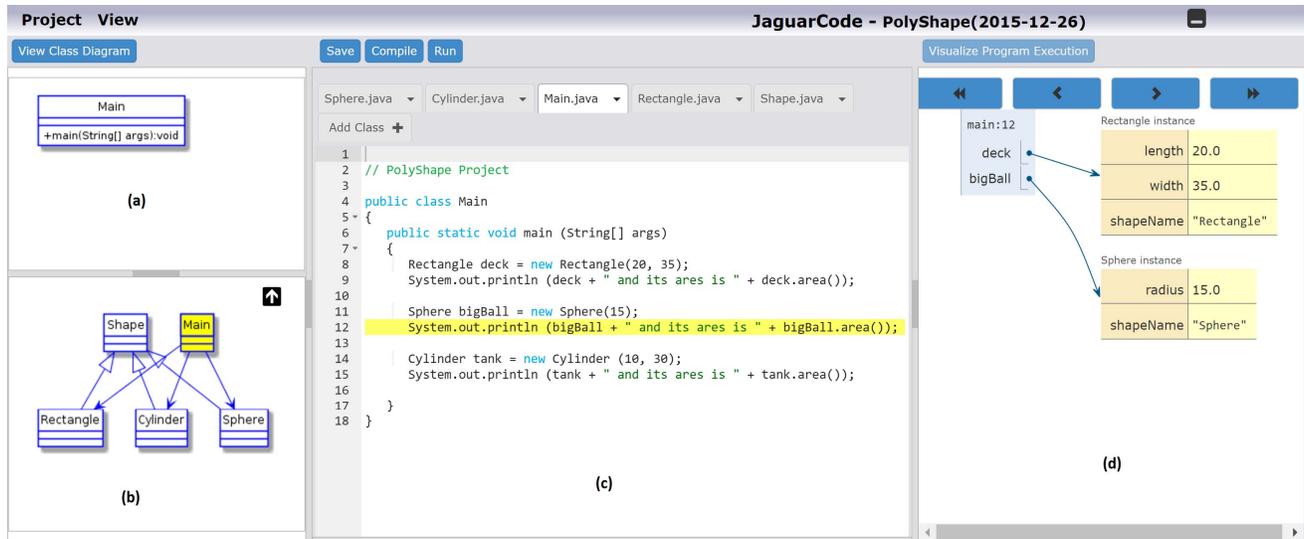


Fig. 1. User Interface of JaguarCode.

Java Visualizer is a web-based program visualization tool [12], which illustrates the dynamic run-time state of a Java program by stepping forwards and backwards through program execution. Java Visualizer re-adapts and uses the Javascript frontend and replaces the backend with Java jail that runs in a sandbox. The backend installation consists of safeexec, a safe execution environment. The safeexec provides a general-purpose sandbox environment which safely executes user programs and prevents any malicious users from causing troubles or mistakes that can damage a server [34]. The Java jail serves as a chroot (changed root) for executing Java programs, and TracePrinter, a Java package, is used to print the traces of Java program executions in JSON format as they execute [35]. The original Java Visualizer is customized and integrated to the JaguarCode interface: only one highlighted bar in yellow is used to indicate the line that has just been executed and the bar jumps to another Java class in the editor as the execution moves on, and multiple Java files made from the editor for a project are merged into a single class file in memory for compilation and execution.

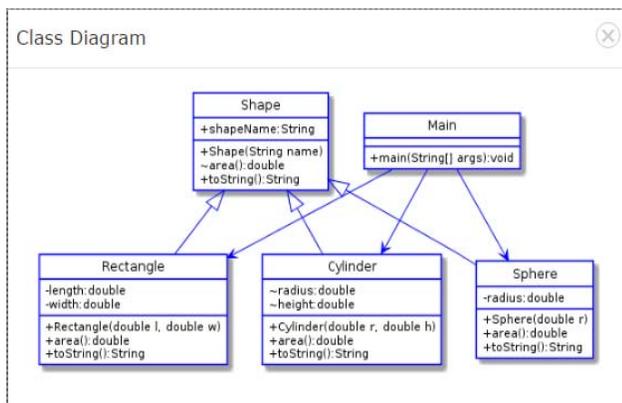


Fig. 2. Enlarged UML Class Diagram.

D. Platform and Target Users

The biggest benefit of using this web-based learning technology is that users simply use a web browser to run JaguarCode, with no required software and plugin installation or configuration on a local computer. There would be no need to know how to install tools or modify environment variables and to carry any kind of storage system to keep and manage their project files. This will provide an easier and more convenient environment for students to do programming on their own schedule with the devices that they use most often.

The target users of JaguarCode are students, learning how to program in Java and how to design an OO system, and instructors, teaching Java programming and designing a system. However, not only students and instructors can benefit from this system but also developers can benefit in tracing and understanding their code, especially for debugging.

IV. EVALUATION

In order to evaluate the learning effectiveness, and user satisfaction of JaguarCode, both quantitative and qualitative experiments were conducted during the spring semester of 2016. The evaluation approach was adopted based on the suggestions by Rubin and Chisnell [29] and the ITiCSE working group [30].

A. Quantitative Evaluation

Two controlled experiments were conducted to investigate the impact of JaguarCode with visualizations. The evaluation measured the results of data on performance from a group of users using JaguarCode supporting static and dynamic representations of source code and a group of users using a standard IDE, NetBeans, with only the source code. Two projects at different difficulty levels were used, i.e., one for a relatively easy project and the other for a relatively hard project while both incorporate fundamental OO concepts such as inheritance and polymorphism.

TABLE I. POLYSHAPE PROJECT QUESTIONNAIRE USED IN BOTH EXPERIMENTS

| Question | Choice |
|--|--|
| 1. What is the parent (super) class of the class P? | a. C b. M c. R d. S |
| 2. How many child (sub) classes does class S have? | b. 1 b. 2 c. 3 d. 4 |
| 3. Can an instance (object) of class R be an instance (object) of class S? | c. Yes b. No |
| 4. Can an instance (object) of class S be an instance (object) of class C? | a. Yes b. No |
| 5. Which method does demonstrate polymorphic behavior? | a. a b. sl c. gr d. sh |
| 6. What is output by the statement at line # 9 marked // Problem 1 in Main.java? | a. R: l is 2.0 and w is 3.0 b. P: l is 2.0 and w is 3.0 c. C: l is 2.0 and w is 3.0 d. S: l is 2.0 and w is 3.0 |
| 7. What is the output by the statement at line # 11 marked // Problem 2 in Main.java? | a. Its a is 1.0 b. Its a is 6.0 c. Its a is 2.0 d. Its a is 3.0 |
| 8. What is the output by the statement at line # 13 marked // Problem 3 in Main.java? | a. R: r is 1.0 b. C: r is 1.0 c. P: r is 1.0 d. S: r is 1.0 |
| 9. What is the output by the statement at line # 15 marked // Problem 4 in Main.java? | a. Its a is 2.0 b. Its a is 3.0 c. Its a is 1.0 d. Its a is 6.0 |
| 10. What is the output by the statement at line # 17 marked // Problem 5 in Main.java? | a. P: r is 1.0 b. C: r is 1.0 and h is 3.0 c. R: r is 2.0 and h is 3.0 d. S: r is 1.0 and h is 3.0 |
| 11. What is the output by the statement at line #19 marked // Problem 6 in Main.java? | a. Its a is 1.0 b. Its a is 2.0 c. Its a is 6.0 d. Its a is 3.0 |

TABLE II. YO-YO PROJECT QUESTIONNAIRE USED IN BOTH EXPERIMENTS

| Question | Choice |
|---|--|
| 1. What is the output by the statement at line # 7 marked // Problem 1 in Main.java? | a. John's payment is 10.0 b. John's payment is 100.0 c. John's payment is 0.0 d. John's payment is 150.0 |
| 2. What is the output by the statement at line # 11 marked // Problem 2 in Main.java? | a. Mary's payment is 10.0 b. Mary's payment is 15.0 c. Mary's payment is 150.0 d. Mary's payment is 0.0 |
| 3. What is the output by the statement at line # 15 marked // Problem 3 in Main.java? | a. Henry's payment is 20.0 b. Henry's payment is 200.0 c. Henry's payment is 0.0 d. Henry's payment is 10.0 |

1) Experimental Design

Hypotheses: Null and alternative hypotheses were accordingly formulated in the following:

H1₀: Having both static and dynamic visualizations available in JaguarCode does not impact the time for understanding programming problems.

H2₀: Having both static and dynamic visualizations available in JaguarCode does not impact the correctness of understanding problems.

H1: Having both static and dynamic visualizations available in JaguarCode reduces the time for understanding programming problems.

H2: Having both static and dynamic visualizations available in JaguarCode increases the correctness of understanding programming problems.

Questionnaire: For the first session of both experiments, five Java classes including Main were used and the questionnaire (TABLE I) was formed for tasks on OO program tracing and understanding. The Java classes simulate a PloyShape project that introduces fundamental inheritance hierarchy with a Shape (S), Rectangle (R), Sphere (P), and Cylinder (C) class. The later three classes inherit variables and methods from their parent class and have an overridden method that calculates the area for each. This is considered as a relatively easy project to understand. For the second session of the experiments, four Java classes including Main were used and the questionnaire (TABLE II) was formed. Three Java classes (Employee, studentEmployee, staffEmployee) simulate the yo-yo effect that causes problems and results in a data flow anomaly from method overriding and polymorphism with an overridden method [23]. This is considered as a relatively difficult project to understand and it serves as an

example of the issues students encounter when transitioning to practical programming applications.

Participants: 16 lower-division computer science majors enrolled in Data Structures and Algorithms at Texas A&M University-Kingsville participated in Experiment 1. The student participants were considered novices without much experience in JAVA except for taking Object-Oriented Software Engineering as a prerequisite of the current course. 75 graduate level computer science majors enrolled in Mobile Application Programming at the same university participated in Experiment 2. The participants were considered as relative experts in the experimental task. To treat the participants in accordance with the "Ethical Principles of Psychologists and Code of Conductor" [24], they were given a small amount of extra credits for the participation toward their final grade in their course.

Method and Procedure: Student participants were divided into two groups. One controlled group was given Java projects (PolyShape project for Session 1 and yo-yo problem project for Session 2) in plain text with NetBeans IDE while the other experimental group was given the same projects with the UML class diagrams and run time visualization of program execution in JaguarCode. In both cases, the participants were instructed to take their time as much as possible to fully understand the given code and be ready to answer questions. To equally balance two groups, the selection of the participants was based on their cumulated grade for the course. Both experiments were held in the computer classrooms located in the College of Engineering at the University. Each experiment had two one-day sessions (one session each day) for each group. Each session lasted approximately one hour.

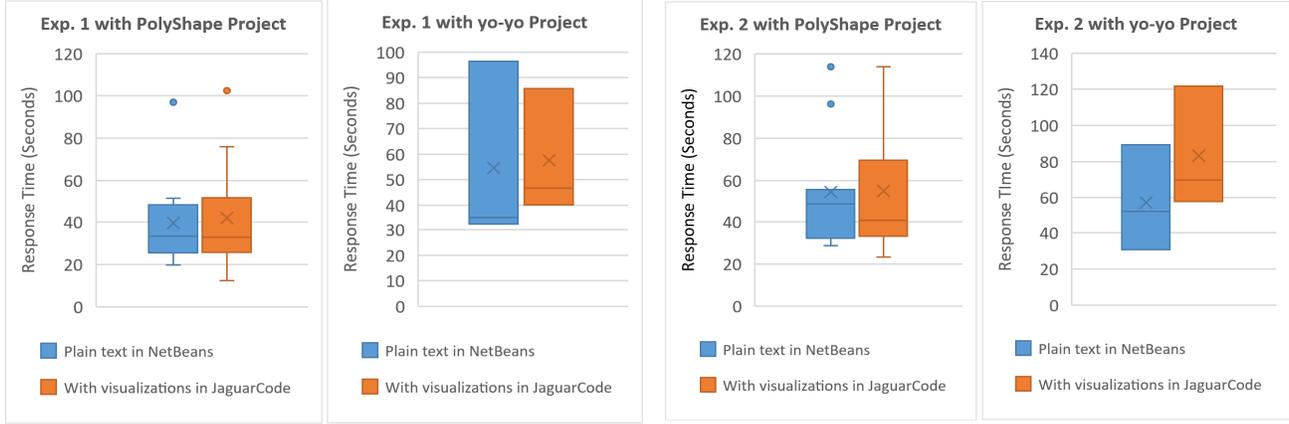


Fig. 3. Box Plots Comparing Average Response Time.

Data Collection and Analysis: To record a response time and answer accurately for each question, the questionnaire was presented in a series of web pages. Each page contained a single question. Whenever a question was answered by selecting one of the choices, the response time and response to that question were saved in a database. A response time was calculated as the time elapsed from when the current question was loaded until the student submitted a response by clicking on the ‘Next’ button for a next question.

The experiments were designed in such a way that each observation (question) in one population (a controlled group) is matched with an observation in other population (an experimental group). The matching is conducted by using the same set of questions for each group. To statistically verify whether both static and dynamic visualizations provided in JaguarCode have impact on the response time and correctness to answer questions, the null hypotheses were tested using the Student’s t-test. In the t-test, differences among the means of both response time and correctness between two populations were studied. To validate that the t-test can be used, the Kolmogorov-Smirnov (K-S) test was applied to verify normal distribution in the sample. As shown in TABLES III and IV, p-values of the K-S test are greater than a value 0.05, which is what we are looking for and is significant that the sample is normal. Due to a small sample size (observation), the two-sample K-S test was applied for the yo-yo problem project.

2) Results of Response Time

The average response time is the total average time taken to respond each question. Our null hypothesis was that “Having both static and dynamic visualizations available in JaguarCode does not impact the time for understanding programming problems”. Box plots in Fig. 3 illustrate the

comparison of the average response times taken to answer questions related to the projects by the two groups (Group 1: the controlled group with source code in plain text using NetBeans IDE and Group 2: the experimental group with the same code along with visualizations using JaguarCode).

In Experiment 1, while average response times (41.942 and 57.507 for the PolyShape and yo-yo projects respectively) in group 2 are slightly higher than those (39.593 and 54.499) in group 1, as shown in TABLE III, the statistical analysis reveals that there is no significant difference between two groups for both projects. In Experiment 2, for the PolyShape project, there is no significant difference between two groups. However, for the yo-yo problem project, the statistical analysis reveals that there is a significant difference between two groups (57.330 and 83.124 for groups 1 & 2 with p-value 0.012). The t-test rejects the null hypothesis, which means that the response time is statistically significantly increased by the availability of visualizations in JaguarCode. Although it does not accept the first alternative hypothesis, this is an interesting finding of an opposite result of what was expected. The alternative hypothesis expected a reduced time for answering questions, but it seemed that students fully utilized the aspects of visual notations in tracing and understanding the code and answering the questions.

3) Results of Correctness

The correctness is the percentage of correct responses to each question. Our null hypothesis was that “Having both static and dynamic visualizations available in JaguarCode does not impact correctness of understanding problems”. Box plots in Fig. 4 show the comparison of the correctness to answer questions related to the projects by both groups.

TABLE III. STATISTICAL ANALYSIS OF RESPONSE TIME

| Session | Observation | Group | Experiment 1 | | | | | | Experiment 2 | | | | | |
|---------|-------------|-------|--------------|---------------|--------|------|--------|-------|--------------|---------------|--------|-------|--------|--------------|
| | | | N | Mean | S.D. | K-S | t | p | N | Mean | S.D. | K-S | t | p |
| 1 | 11 | 1 | 6 | 39.593 | 21.340 | 0.14 | -0.378 | 0.357 | 33 | 54.398 | 26.971 | 0.08 | -0.118 | 0.454 |
| | | 2 | 10 | 41.942 | 26.676 | 0.27 | | | 42 | 54.886 | 31.587 | 0.46 | | |
| 2 | 3 | 1 | 6 | 54.499 | 36.342 | 0.58 | -0.430 | 0.354 | 33 | 57.330 | 29.714 | 0.518 | -6.155 | 0.012 |
| | | 2 | 10 | 57.507 | 24.586 | | | | 42 | 83.124 | 33.860 | | | |

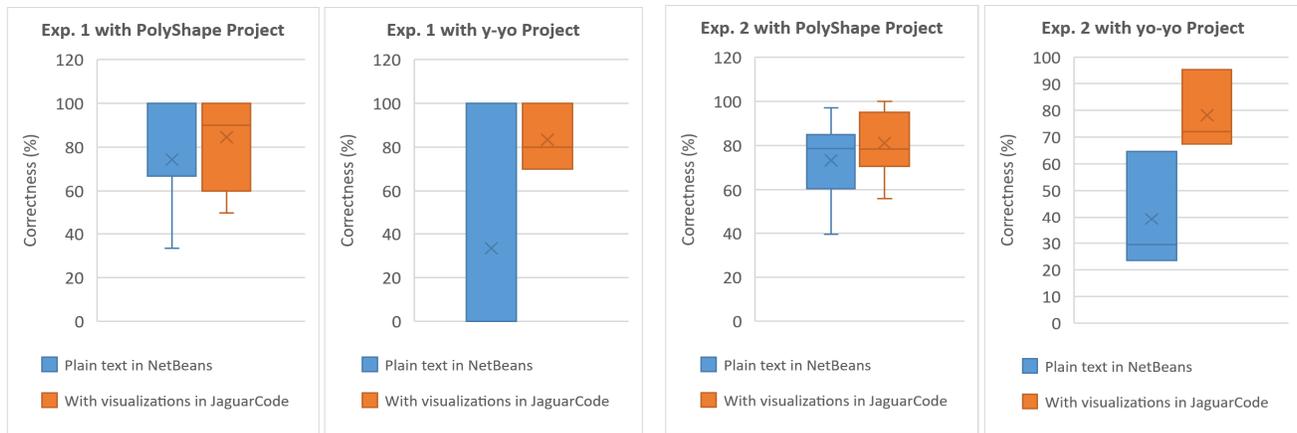


Fig. 4. Box Plots Comparing Correctness.

In Experiment 1, for the PolyShape project, the statistical analysis reveals that there is a significant difference between two groups (means 74.243 and 84.545 for groups 1 & 2 with p-value 0.025 as in TABLE IV). The result of t-test rejects the null hypothesis, which means that the correctness is statistically significantly increased by the availability of visualizations. For the yo-yo problem project, although the correctness for all three questions (100%: mean 83.333) in group 2 are equal to or higher (mean 33.333) than those in group 1, there is no significant difference between two groups (p-value 0.093). This is due to higher variances with a relatively smaller sample size (observation 3).

For Experiment 2, as the correctness (means 81.192 and 78.293 for the PolyShape and yo-yo projects respectively) in group 2 are higher than those (73.281 and 39.127) in group 1, as shown in TABLE IV, statistical analysis reveals that there is significant difference between two groups for both projects. The results of t-test reject the null hypothesis and accept the alternative hypothesis, meaning that the correctness is statistically significantly increased by the availability of visualizations in JaguarCode as the p-value is with 0.027 for the PloyShpe project and 0.006 for the yo-yo problem project.

4) Discussion

In summary, it was observed that students in the experimental group using two aspects of visualizations in JaguarCode consistently performed better to correctly answer questions on program understanding for both easy and hard projects than the controlled group. Therefore, the statistical analysis of the experimental data supports the conclusion that having both static and dynamic visualizations available in JaguarCode does positively impact on increasing the correctness of program understanding and tracing problems, in

particular, for relatively difficult questions.

B. Qualitative Evaluation

1) Experimental Design

The qualitative study was designed to access whether using JaguarCode with the static and dynamic visualization could contribute to its goals of meeting user's needs and providing satisfaction. In this study, questionnaires with System Usability Scale (SUS) suggested by ISO standard 9241 [31] were used to gather data on how satisfied users were with JaguarCode. The degree of disagreement or agreement on a 5-point rating scale was used that ranged from 1 'Strongly Disagree' to 5 'Strongly Agree'.

Objectives: The objectives of this evaluation study were established: a) Do the UML class diagrams in JaguarCode support student's understanding of object oriented concepts? b) Does the run-time visualization in JaguarCode support student's understanding of object-oriented programming? c) Does JaguarCode make learning of object-oriented program easier? d) Is JaguarCode easy to use? e) Are students satisfied and comfortable using JaguarCode?

Participants: 41 students (6 undergraduates and 35 graduates) out of 52 students from the previous experimental groups participated in this evaluation. They experienced JaguarCode through two sessions of the experiments for about one and half hours before taking the questionnaire. Among the 41 respondents, the majority, 29 (71%) of respondents indicated that they were either an advanced beginner or competent in rating themselves in Java. All respondents were Computer Science majors: including 35 graduates, 4 seniors, and 2 sophomores.

TABLE IV. STATISTICAL ANALYSIS OF CORRECTNESS

| Session | Observation | Group | Experiment 1 | | | | | | Experiment 2 | | | | | |
|---------|-------------|-------|--------------|---------------|--------|-------|--------|--------------|--------------|---------------|--------|------|--------|--------------|
| | | | N | Mean | S.D. | K-S | t | p | N | Mean | S.D. | K-S | t | p |
| 1 | 11 | 1 | 6 | 74.243 | 21.556 | 0.62 | -2.232 | 0.025 | 33 | 73.281 | 16.915 | 0.93 | -2.190 | 0.027 |
| | | 2 | 10 | 84.545 | 20.671 | 0.15 | | 42 | | | | | | |
| 2 | 3 | 1 | 6 | 33.333 | 57.735 | 0.518 | -1.987 | 0.093 | 33 | 39.127 | 22.273 | 0.1 | -9.231 | 0.006 |
| | | 2 | 10 | 83.333 | 15.275 | | | 42 | | | | | | |

TABLE V. MEAN RATING AND PERCENT AGREEMENT FOR VISUALIZATION RELATED QUESTIONS

| Associated objective | Question | Strongly Disagree (1) | 2 | 3 | 4 | Strongly Agree (5) | Mean Rating | *Percent Agree |
|----------------------|---|-----------------------|---|---|----|--------------------|-------------|----------------|
| a | 1. UML class diagrams helped me better understand overall structure of Java programs. | 0 | 1 | 0 | 20 | 20 | 4.439 | 97.56% |
| a | 2. UML class diagrams help me to better understand Object-Oriented design concepts. | 0 | 0 | 5 | 15 | 21 | 4.390 | 87.80% |
| b | 3. I understand the “dynamic run time visualization of Java program execution”. | 0 | 0 | 1 | 14 | 26 | 4.610 | 97.56% |
| b | 4. The “dynamic run time visualization of Java program execution” helps me correct and improve the quality of my program. | 0 | 0 | 6 | 13 | 22 | 4.392 | 85.37% |
| c | 5. Synchronized UML class diagram and run time visualization along with source code make it easier for me to comprehend Java program. | 0 | 2 | 1 | 13 | 25 | 4.488 | 92.68% |
| c | 6. Synchronized UML class diagram and run time visualization along with source code alleviate the intimidation of Java programming. | 0 | 1 | 8 | 11 | 21 | 4.268 | 78.05% |

*Percent Agree (%) = Strongly Agree (5) and Agree (4) responses combined.

TABLE VI. MEAN RATING AND PERCENT AGREEMENT FOR USABILITY RELATED QUESTIONS

| Associated objective | Question | Strongly Disagree (1) | 2 | 3 | 4 | Strongly Agree (5) | Mean Rating | *Percent Agree |
|----------------------|---|-----------------------|---|---|----|--------------------|-------------|----------------|
| d | 1. Jaguar Code makes it easier for me to start writing Java programs | 0 | 0 | 6 | 12 | 23 | 4.415 | 85.37% |
| e | 2. The interface of Jaguar Code is user friendly. | 0 | 0 | 6 | 9 | 26 | 4.488 | 85.37% |
| d | 3. Jaguar Code is easy to use. | 0 | 0 | 3 | 7 | 31 | 4.683 | 92.68% |
| e | 4. I enjoy the time I spent using Jaguar Code. | 0 | 1 | 4 | 9 | 27 | 4.512 | 87.80% |
| e | 5. Working with Jaguar Code is satisfying. | 0 | 1 | 5 | 7 | 28 | 4.512 | 85.37% |
| d | 6. The way that Jaguar Code is presented is clear and understandable. | 0 | 0 | 3 | 12 | 26 | 4.561 | 92.68% |
| e | 7. I was comfortable in programming with Jaguar Code. | 0 | 0 | 3 | 13 | 25 | 4.537 | 92.68% |
| e | 8. I would like to use Jaguar Code for the rest of the subject. | 0 | 1 | 2 | 16 | 22 | 4.439 | 92.68% |
| e | 9. The speed of using Jaguar Code is fair enough. | 0 | 0 | 4 | 13 | 24 | 4.488 | 90.24% |

*Percent Agree (%) = Strongly Agree (5) and Agree (4) responses combined.

Method and Procedure: In both Experiments, after completing the source code related questions in session 2, a series of visualization and usability related questions described in TABLES V and VI were asked for the participants to rate the degree of disagreement or agreement on how satisfied they were with JaguarCode. The questions were also presented on web pages and responses were saved in the database.

2) Results of Visualization Related Questions

For the UML class diagram related questions (1 & 2 in TABLE V), 40 (97.56%) and 36 (87.80%) respondents indicated that they strongly agreed or agreed that the UML class diagrams helped them better understand overall structure of Java programs (mean rating = 4.439) and Object-Oriented design concepts (mean rating = 4.390). For the run time visualization related questions (3 & 4 in TABLE V), 40 (97.56%) and 35 (85.37%) respondents indicated that they strongly agreed or agreed that they understood the “dynamic run time visualization of Java program execution provided in JaguarCode (mean rating = 4.610) and it helped them correct and improve the quality of their program (mean rating = 4.392). For the two aspects of synchronized static and

dynamic visualization related questions (5 & 6 in TABLE V), 38 (92.68%) and 33 (78.05%) respondents indicated that they strongly agreed or agreed that both visualizations together make it easier for them to write Java programs (mean rating = 4.488) and both visualizations together could alleviate the intimidation of Java programming (mean rating = 4.268).

3) Results of Usability Related Questions

TABLE VI shows the mean rating and percentage agreement for each of the usability related questions. The mean ratings are high and consistently ranging between 4.415 and 4.683 along with high percentage of agreements (85.37% and 92.68%) throughout all questions. The respondents strongly agreed or agreed that JaguarCode makes it easier for them to start writing Java programs (mean rating = 4.415), its user interface is user friendly (mean rating 4.488), it was easy to use (mean rating = 4.683), they enjoyed the time spent using JaguarCode (mean rating = 4.512), working with JaguarCode was satisfying (mean rating = 4.512), the way that JaguarCode is presented was clear and understandable (mean rating = 4.561), and they were comfortable in programming with JaguarCode (mean rating = 4.537).

TABLE VII. MEAN RATING AND PERCENT AGREEMENT FOR OBJECTIVES

| Objective | Associated Questions | Total | Strongly Disagree (1) | 2 | 3 | 4 | Strongly Agree (5) | Mean Rating | *Percent Agree |
|-----------|--|-------|-----------------------|---|----|----|--------------------|-------------|----------------|
| a | Visualization related questions 1 & 2 | 82 | 0 | 1 | 5 | 35 | 41 | 4.415 | 92.68% |
| b | Visualization related questions 4 & 5 | 82 | 0 | 0 | 7 | 27 | 48 | 4.501 | 91.46% |
| c | Visualization related questions 7 & 8 | 82 | 0 | 3 | 9 | 24 | 46 | 4.378 | 87.5% |
| d | Usability related questions 1, 3, & 6 | 123 | 0 | 0 | 12 | 31 | 80 | 4.553 | 91.74% |
| e | Usability related questions 2, 4, 5, 7, 8, & 9 | 246 | 0 | 3 | 24 | 67 | 152 | 4.496 | 89.02% |

*Percent Agree (%) = Strongly Agree (5) and Agree (4) responses combined.

4) For Associated Objectives

For the associated objectives, the mean ratings of all objectives remain high and consistently ranging between 4.378 and 4.553 (TABLE VII). With percentage agreements ranging between 87.5% and 92.68%, the respondents strongly agreed or agreed that the UML class diagrams in JaguarCode helped them understand object-oriented concepts (mean rating = 4.415). The run-time visualization in JaguarCode also supported their understanding of object-oriented programming (mean rating = 4.501), made their learning of object oriented program easier (mean rating = 4.378), was easy to use (mean rating = 4.553), and they were satisfied and comfortable using JaguarCode system (mean rating = 4.496).

5) Discussion

Overall, student respondents were satisfied with the aspects of static visualization in UML class diagrams and the run time visualization of program execution provided in JaguarCode. Their key comment of using JaguarCode was that there is no installation requirement, which they can use on any device to compile and test programs. It was also mentioned that the visualizations helped them understand the code, the flow of program and the structure of inheritance and classes. For any improvement to be made on JaguarCode, a few respondents suggested that specifying errors during editing and after executing programs can be improved, and the execution speed must also be improved. The support of other programming languages was also suggested.

C. Threats to Validity

The evaluation study was implemented for a short period of time with a relatively small subjects. Further study with a larger group of participants may be needed to generalize the evaluation. Other weak points of the experiments are related to the scale and domain of the programs used, the types of the questions asked, and the classification of novice and expert users. Further experiments involving large-scale programs, more specific OO related questions, and equally balanced number of subjects would be carefully designed. In a sense that an educational programming environment better supports student performance compared with environments without visual representations of code, there would be additional empirical studies in a systematic comparison with other educational IDEs.

V. CONCLUSION

This paper has reported on an initial evaluation of JaguarCode in terms of its effectiveness and user satisfaction

through quantitative and qualitative experiments. The quantitative evaluation study did explore differences in correctness and time usage of program understanding and tracing problems. The results of the experiments support the conclusion that students in the experimental group using two aspects of visualizations in JaguarCode performed better to questions on program tracing and understanding than the controlled group. The application of t-tests rejects the second null hypothesis meaning the correctness is significantly increased by the availability of visualizations along in JaguarCode. With regard to the response time, the statistical analysis from Experiment 2 reveals that, for the relatively hard project, there is a significant difference between the controlled and experimental groups. While the result of t-test does not accept the alternative hypothesis, it rejects the first null hypothesis, which means that the response time is significantly increased by the availability of visualizations. This is an interesting finding of how both visualizations did affect students' understanding on program execution. Students took longer to answer, in particular, the relatively difficult questions using the visualizations provided in JaguarCode, which led to higher accuracy in answering the questions correctly.

In the qualitative evaluation, student feedback on the usability of JaguarCode interface was evaluated to investigate whether the JaguarCode environment helps students make their OOP learning easier and help their understanding of OO concepts, and whether the interface would contribute to providing satisfaction. The results of the evaluation support the positive effect of JaguarCode on helping students better understand OO concepts and meeting goals of providing comfortability and satisfaction for both the observations of visualization and usability related questions and the associated objectives. Regarding any difficulties or improvements to be made in JaguarCode, a few respondents have identified that the speed of visualizing was a little slow and error messages during editing and executing programs were not informatively displayed. Therefore, providing detailed error messages and visualization speed must be improved. On specific benefits of using JaguarCode, it was clearly observed that users were satisfied using JaguarCode - it is simple, user friendly, understandable, and easy to use.

VI. FUTURE WORK

Important goals for future work of JaguarCode are to extend its use and embed more functionality beyond the current version. Plans are to include a Virtual Tutoring (VT) capability, learning data analytics, and simplified run time visualization of program execution with the improved speed.

The VT system will consist of two major components, a screen sharing capability by presenting a student's code in real time to a course instructor or a teaching assistant (TA) and a list of commonly used coding segments. The student will be provided a mentoring/tutoring session room to communicate with the instructor or TA who can instantly join the room and connect to the code editor to help them complete an assignment. The student will also be provided a list of code segments from which to make a selection. The selected segment can be imported to the student's code at the cursor's current position in the editor window. The objectives of the VT system are to give students real-time assistance on coding and an easier means of developing code segments that are unfamiliar to them.

Student's programming patterns can be saved in log files to measure, collect, analyze, and report of data about their behavioral patterns in programming. The purpose of student's learning analytics is to enhance student's programming, logical reasoning, and thinking skills through individualized mentoring based on individual learning speed and ability. As indicated in the evaluation, the improvement that must be made to JaguarCode is the speed of visualizing program execution. A sandbox environment to safely execute Java programs on the server causes the slow speed of the visualization. A better performance in speed must be achieved with careful design consideration.

REFERENCES

- [1] I. Lavy, R. Rashkovits, and R. Kouris, "Coping with abstraction in object orientation with a special focus on interface classes," *Computer Science Education*, 19 (3):155-177, 2009.
- [2] N. Liberman, C. Beeri, and Y. Ben-David Kolikant, "Difficulties in learning inheritance and polymorphism," *Trans. Computer Education*, 11(1):4:1-4:23, Feb. 2011.
- [3] Nghi Truong, Peter Bancroft, Paul Roe, "A Web Based Environment for Learning to Program," *ACSC '03 Proceedings of the 26th Australasian computer science conference - Volume 16*, pp. 255-264.
- [4] Dianne Hagan and Selby Markham, "Teaching Java with the BlueJ Environment," *Proceedings of Australasian Society for Computers in Learning in Tertiary Education Conference ASCILITE*, 2000.
- [5] Philip J. Guo, "Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education," *Proceedings of the ACM Technical Symposium on Computer Science Education*, March 6-9, 2013.
- [6] Noa Ragonis and Mordechai Ben-Ari, "On Understanding the Statics and Dynamics of Object-Oriented Programs," *SIGCSE '05*, February 23-27, 2005, St. Louis, Missouri, USA.
- [7] Jeong Yang, Young Lee, and David Hicks, "Synchronized Static and Dynamic Visualization in a Web-Based Programming Environment," *IEEE International Conference on Program Comprehension (ICPC)*, May 16-17, 2016.
- [8] Jeong Yang, "JavelinaCode: A Web-Based Object-Oriented Programming Environment with Static and Dynamic Visualization," Ph.D. Dissertation, Auburn University, 2016.
- [9] Brandon Earwood, Jeong Yang, and Young Lee, "Impact of Static and Dynamic Visualization in Improving Object-Oriented Programming Concepts," *IEEE Frontiers in Education 2016: The Crossroads of Engineering and Business*, October 12-15, 2016.
- [10] Michael Kölling, "The design of an object-oriented environment and language for teaching," PhD Dissertation, University of Sydney, 1999.
- [11] Van Haaster, K. and Hagan, D., "Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool." *Information Science & Information Technology Education Joint Conference*, Rockhampton, QLD, Australia, June, 2004.
- [12] Jeong Yang, Young Lee, David Hicks, and Kai Chang, "Enhancing Object-Oriented Programming Education using Static and Dynamic Visualization," *2015 IEEE Frontiers in Education (FIE): Launching a New Vision in Education Engineering*, pp. 806-810, 2015.
- [13] Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari, "Visualizing Programs with Jeliot 3," *Proceedings of the International Working Conference on Advanced Visual Interface AVI*, May 2004.
- [14] Andrés Moreno and Mike S. Joy, "Jeliot 3 in a Demanding Educational Setting", *Fourth International Program Visualization Workshop*, June 29-30, 2006.
- [15] J. H. Cross, T. D. Hendrix, D. A. Umphress, L. A. Barowski, J. Jian, and L. Montgomery, "Robust Generation of Dynamic Data Structure Visualizations with Multiple Interaction Approaches," *ACM Transactions on Computing Education*, vol. 9, no. 2, pp. 13:1-13:32, June 2009.
- [16] J. Jian, J. H. Cross, T. D. Hendrix, and L. A. Barowski, "Experimental Evaluation of Animated-Verifying Object Viewers for Java," in *Proceedings of SoftVis 2006*, Brighton, UK, 2006.
- [17] J. H. Cross, T. D. Hendrix, L. A. Barowski, and D. A. Umphress, "Dynamic Program Visualizations - An Experience Report," *Proceedings SIGCSE 2014*, pp. 609-614, 2014.
- [18] Andre L. Santos, "AGUIA/J: A Tool for Interactive Experimentation of Objects," *ACM ITiCSE'11*, June 27-29, 2011, Darmstadt, Germany.
- [19] Andre L. Santos, "Novel Interaction Metaphors for Object-Oriented Programming Concepts," *14th International Conference on Computer Science Education*, Koli, Finland, 2014.
- [20] Paul V. Gestwicki, Bharat Jayaraman, "JIVE: Java Interactive Visualization Environment," In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 226-228, 2004.
- [21] Paul V. Gestwicki, Bharat Jayaraman, "Methodology and Architecture of JIVE," In *Proceedings of the 2005 ACM Symposium on Software Visualization (SOFTVIS)*, pp. 95-104, 2005.
- [22] Bharat Jayaraman, Charlotte M. Baltus, "Visualizing Program Execution," In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pp. 30-37, 1996.
- [23] Offutt, J., Alexander, R., Wu, Y., Xiao, Q., Hutchinson, C. "A fault model for subtype inheritance and polymorphism," In: *Proceedings of the 12th International Symposium on Software Reliability Engineering*, pp. 84-93. *ISSRE IEEE Computer Society*, Washington, DC (2001).
- [24] *Ethical Principles of Psychologists and Code of Conduct*, <http://www.apa.org/ethics/code/>.
- [25] Ace, The High Performance Code Editor for the Web, <http://ace.c9.io/>.
- [26] PlantUML, <http://plantuml.com/>.
- [27] Java Visualizer, http://www.cs.princeton.edu/~cos126/java_visualize/.
- [28] Online Python Tutor, <http://www.pythontutor.com/>.
- [29] Jeffrey Rubin and Dana Chisnell, *Handbook of Usability Testing, Second Edition: How to Plan, Design, and Conduct Effective Tests*, Wiley Publishing, Inc., 2008.
- [30] Almstrum, V. L., Dale, N., Berglund, A., Granger, M., Currie Little, J., Miller, D. M., et al., "Evaluation: Turning technology from toy to tool: Report of the Working Group on Evaluation," In *ITiCSE '96: Proceedings of the 1st conference on integrating technology into computer science education*, Volume 28 Issue SI, 1996.
- [31] ISO 9241-210:2010(en), *Ergonomics of human-system interaction - Part 210: Human-centered design for interactive systems*.
- [32] Madhusudan Srinivasan, Young Lee, and Jeong Yang, "Enhancing Object-Oriented Programming Comprehension using Optimized Sequence Diagram," *IEEE Conference on Software Engineering Education and Training (ICSEE)*, pp. 81-85, 2016.
- [33] Madhusudan Srinivasan, Jeong Yang, and Young Lee, "Case Studies of Optimized Sequence Diagram for Program Comprehension," *International Conference on Program Comprehension (ICPC)*, 2016.
- [34] Safe Execution Environment, <https://github.com/cemc/safeexec>.
- [35] Java Jail, https://github.com/daveagp/java_jail.