Non-Computer Science Majored Women Students Perspective on a Pictorial Programming Environment

Shannon Sillessi, Cihan Varol, and Hacer Varol

***Abstract*—**Although, programming classes have been taught with variety of ways and different tools to increase the student retention, women and minorities are still underrepresented in computer science and the awarded degree numbers are still way behind from the 2000 - 2006 numbers. In order to address this problem, a drag-on-drop programming environment is created to teach introductory level object oriented programming course for the students who have no prior programming experience. The framework eliminates one of the main challenges, if not the most, namely syntax issues with creating the programming statements automatically. With that way the students focus more on the logical issues of the task, instead of spending time on correcting typos and syntax errors. The applied survey to the student who is majored in another program but taking the Introduction to Programming course revealed that 88 percent of them found the application beneficial. Survey completed by non-Computer Science (CS) majored women students indicates that this new application will increase the interest level in computer science, while 82 percent of the non-CS majored women students prefer to use this application instead of current programming environment**.**

***Keywords*—**Computer Science, Drag-on-Drop, Java Programming, Visual Programming.

I. Introduction

Although, overall the total number of enrolled majored students and awarded degrees in computer science is increasing in the last couple of years, it is still behind the numbers from the period of 2000 and 2006 [1]. Moreover, the 11% female population in computing area is almost half of the all women population in science and engineering field [1]. According to the research done by [2, 3], frustration while programming, math involvement, and spending most of the time in front of the computer are some of the reasons why students don't pursue a degree in the computing area. Besides those, the belief that computer science is a competitive field is also an extra discouraging reason for female students which push them away from computing major. Because of these, the lack of students in the computing area creates a good amount of unfilled jobs in the IT sector where there is dire need to have skilled employees. Therefore, in order to attract more students and increase the retention, different strategies have been used [4, 5]. Some departments offered scholarships [5], companies provided paid internship opportunities and summer trainings [5], and faculty promoted their departments with teaching in different styles and exposing the students with exciting research projects [6, 7, 8, and 9]. The effort put by the faculty varies among the level and style being used, while some of those studies target introductory level computer programming courses [7, 10]. The main rationale behind this is that some students change their majors after taking the first course from computer science [11]. Also, introductory level programming instructors support the fact with claiming the students are having hard time when they take their first programming course. Therefore, in order to promote the computer science field, the introductory level programming courses need to be more appealing to the students rather than being so difficult that they lose interest in this field of study.

Each student's learning style and cycle is different from each other. Therefore, finding and applying one technique and tool in the programming language courses is a challenge. However, based on the reasons that the students change their major from computer science to another field can yield to a framework that will help student retention. Therefore, a new programming editor for Java is created which helps lessen the frustration experienced by beginner programmers. The new editor employs a drag-and-drop feature to improve students' ability to programming while allowing them to grasp main concepts. The editor is designed to generate automatic statements and related syntax according to the programmer's need while minimizing the time spent on fixing the errors.

In order to test the editor, a preliminary survey was conducted on the students in the Programming Fundamentals I course in the first quarter of the semester. After which, we let the students use the editor closer to the end of the semester once they had been programming with a plain text editor. Then we administered a post-test survey to see how they felt about the new editor, what they preferred and what helped them the most. According to the surveys, 88 percent of the non-CS majored women students found the application useful. Moreover, survey completed by non-CS majored women students indicates that this new application will increase the interest level in computer science and 82 percent of them

S. Silessi is with Sam Houston State University, Huntsville, TX 77341 USA (e-mail: sgs008@ shsu.edu).

C. Varol is with Sam Houston State University, Huntsville, TX 77341 USA (corresponding author to provide phone: 936-294-3930; fax: 936-294-4312; e-mail: cvarol@ shsu.edu).

H. Varol is with Sam Houston State University, Huntsville, TX 77341 USA (e-mail: hxv002@ shsu.edu).

prefer to use this editor instead of the current programming environment.

In Section 2, we will describe what has been done previously in the field of Computer Science to address this issue, and what applications have been created to aid novice programmers with learning to program. In Section 3, after evaluating the advantages and disadvantages of the current programming environments, we will describe how our findings contributed to the creation of the new editor called *Dragon Drop*. In section 4, the test case of the editor and results will be revealed and the paper will be finalized with discussion and conclusion section.

II. RELATED WORK

Research in related work has determined several projects were developed to aid students in grasping the concepts and syntax essential to completion of their programming assignments. *Karel Software* was a success at the high school level in helping students transition into the world of coding with a programming tool called *Alice* [12]. *Alice* is a 3D programming environment made for entry-level programmers that allows the users to use scripts to create and control the 3D environment [12]. It's based on the python programming language and uses many of its features in the available scripts to control the 3D objects in the program and interact with input from the user [12]. The built-in commands of the language are separated into two groups: one group moves the objects in the environment in different directions and views, and the other controls the nature of objects destroying and creating [12]. In conventional 3D programming, one must know how to structure the methods, decisions and looping in order to perform these actions [12]. *Alice* uses GUI (Graphical User Interface) controls and gives a visual representation to make operations easier to understand without the user's prior knowledge of 3D programming [12].

Joey C.Y. Cheung, and et. al., proposed another approach for junior high school students called *Bricklayer*. *Bricklayer* is a text-enhanced graphical programming environment designed to stimulate the learning interests of students while also encouraging them to learn both programming logic and syntax [13]. It allows specific syntax statements to be generated immediately where the students can view them as they drag a block to a particular place [13]. This action allows students to instantly know what happens in the coding while they are creating a story or animations [13]. The programming tool is written in *JavaScript* so it can be run over a web browser, and the source code is generated in the *C* language [13]. What makes this programming environment unique is that it requires students to drag and drop the separate 'If,' 'Then' and 'End If' components of a conditional statement individually into the construction area in order to give students a more realistic view of programming concepts. This would make it easier for them to make the switch to conventional textual programming later on [13]. *Bricklayer* also gives the user the ability to copy source code and past it into another text editor for further modification [13].

Among the more popular development environments for programming courses is the mid-weight IDE called *BlueJ*, developed by Michael Kolling and John Rosenberg. *BlueJ* features a graphical class structure display that illustrates the relationships between each class [14]. Using *BlueJ* as a programming tool requires the instructor to be more involved in teaching the students an understanding of the concepts and syntax of programming, but proves to be more effective than the strictly text-based approach [15]. There is, however, no visual representation of objects [14].

Faculty and students at Washington University have developed a programming GUI by the name of *Jpie*, in the pursuit of making programming available to people who would normally quit or not be interested at all by having them focus on actually creating software rather than learning the intricacies of the language that they are using [16]. *Jpie* achieves this goal by making most abstractions and constructs of the language into visible representation for the students to use directly [16]. It uses reflection in *Java* to allow live modifying of the software to get out of the compile-then-build-and-see-if-it-works mentality [16]. *Jpie* uses constructs called 'capsules' which includes all variable declarations, variable accesses, methods, and constructors. Each of these capsules is visually coded to show its type and its color for scope, demonstrating exactly what parts of the program it affects. It also utilizes drag-and-drop operations for "get" and "set" commands [16].

Developed at both Kent and Deakin University with Sun Microsystems' support is *Greenfoot*. *Greenfoot* is an interactive development environment for *Java* created with the purpose to educate high school and undergraduate level students in programming [14]. Using *Greenfoot*, users have the support to develop graphical applications in 2D, particularly games [16]. *Greenfoot* began with Michael Kolling from the *BlueJ* team, and Poul Henriksen [17]. Its integration is an expansion on other existing tools, namely *BlueJ*, for the development environment. *Greenfoot* is a highly visual and interactive tool meant to reduce overall time needed to learn each of the currently available tools that *Greenfoot* is based on [14].

Jeroo is an available narrative tool that provides an integrated development environment, and an animation window in which the student takes the role of protecting animals called Jeroos on an island with dangers than can hurt the animals [18]. The student configures the island and creates programs to define how the Jeroos deal with the challenges of their surroundings. This is a multi-language tool. *Scratch* is another combination of narrative and visual programming that uses drag-and-drop to piece program fragments together like a puzzle [18]. It also allows for students to test out their program while they are coding it. This is useful for more visual thinkers [16].

There are many programming tools that follow a different type of programming tool design [18]. If one is looking for a flow-model tool, they have *RAPTOR*. It runs on the Microsoft Windows operating system only and provides information in the form of handouts on all the stages of programming up to intermediate level [18]. *JHAVE* is a tool used for algorithm visualization that is driven by visual representation of the code and pop-up questions. *Game Maker* has its own proprietary programming language that the students use to create their own game, after first learning about the basics of programming [18]. *Baltie* is a visual and tiered language tool [18]. This program includes an interactive mode and icons for beginners, then advances to a combination of icons and text commands or text by itself as the user develops skill [18]. The environment can be exported into *Visual Studio.NET* and has a large knowledge base of tutorials.

In the pursuit of minimizing the problems with teaching introductory and intermediate programming to fresh new learners, developers have created various tools that all try and tackle the task in their own approach (as shown above). These tools have been categorized as the following: narrative tools, visual programming tools (*Baltie*, *Game Maker, JHAVE*), flow-model tools (*RAPTOR*), specialized output realizations, and tiered language tools (*Baltie*). Narrative tools use a storyboard structure to introduce programming. Visual programming immerses the student in a graphical environment which helps to graphically interact with segments of code, minimizing the trouble with learning language syntax and concentrating on a visual product [18]. Flow-model lets the user develop the structure of a program and its pieces by connecting the elements of the program in more of a “blueprint” fashion [18]. Specialized output realization helps to develop motivation by providing feedback in some kind of multimedia form to reinforce understanding [18]. Lastly, tiered language programming tools allow the user to define their programming skill level in the support tool covering a wide range of programming levels, using more complex features as experience dictates [18].

Each of these tools has its own advantages, and they often have advantages in common. Pertaining to the problem statement, we propose what would be classified as more of a visual programming tool to help the students starting out on programming to have some kind of graphical representation of the inner-workings of a whole program. The problem statement reveals an advantage in and of itself: novice programmers will have a way of learning the basics of programming by learning and implementing the syntax through visual representation. This is the main goal, and previous iterations of a visual programming tool such as *Alice* and *Greenfoot* have proven this approach to be effective in strengthening a student’s grasp of fundamental concepts, programming logic, base to intermediate syntax, and overall level of confidence when using them to create programs [14].

Given the benefits of such tools, they are not without their own issues. One disadvantage with programming tools such as the ones listed above is that a student may end up too reliant on the program and unable to smoothly make the transition to more conventional coding, or for that matter, even another programming tool. It is possible to go too far with the visualization and make it too complex for entry level students to understand them. Also, depending on the implementation, it may be difficult to trace errors. Since part of the reason for visual programming is to streamline the learning process and have students create real, functional programs, it decreases the margin for error. Students need to understand where an error might be and how to fix it in an early stage if they hope to advance to more challenging programming assignments. Specifically, *Alice* has a problem with moving from closed loop animation to an interactive loop’s response variable, as the learning curve for that particular obstacle in syntax is challenging for students who use it [19]. Some programs also do not have the ability to display source code dynamically as the user makes changes graphically, which can cause a hang-up in transition from visual programming to conventional text-based programming.

What seems to be a common approach among the architecture of visual programming tools is first and foremost, a friendly and interactive interface. Accompanying the interface is a way to display source code that reflects what the user is doing graphically. Furthermore, users should be able to create, resume, and save their work in a fashion that will allow them to compile and run their program in a way that resembles conventional programming, so not to create too much of a gap between visual and text-based. The beneficial features of the visual elements must in some way identify and communicate to the user which statements, methods, classes, or other syntax they are manipulating- whether it be through icons, colors, animations, audio, or whatever method is chosen. The programming tool should provide easy manipulation of each of these pieces of code, but given constraints on what the correct coding requires with no errors. For example, the tool can provide a message explaining that in order to use an object (or variable), one must define it. Lastly, given the history of past projects, the programming tool should be versatile. The features of the tool are meant to help the user through the learning process of the concepts and syntax that is required in a beginner’s programming course, and some to intermediate and higher. These features should be able to scale for program complexity.

III. Methodology

The goals of creating *Dragon drop* were aimed at the students taking the Programming Fundamentals - I class.

Our goals are

- Minimize Syntax errors
- Ease of transition
- Show the immediate use

The diagram shown in Figure 1 illustrates the high level architecture diagram of *Dragon Drop*. The JavaAppUI class extends the predefined *Java* class JFrame. The JFrame class contains the methods to implement a GUI window with a title, border, menu bar, and user-specified components. The JavaAppUI class is the main class of the application and contains all the methods implementing the drag and drop features. The JavaAppUI class creates an object of the SourceString subclass, which extends the predefined *Java* class SimpleJavaFileObject. Since the super constructor cannot be invoked on the SimpleJavaFileObject class due to it being private, the SourceString class is used to create a constructor that overrides the private constructor in the

SimpleJavaFileObject class. Together, these classes allow for dynamic compiling of *Java* files.

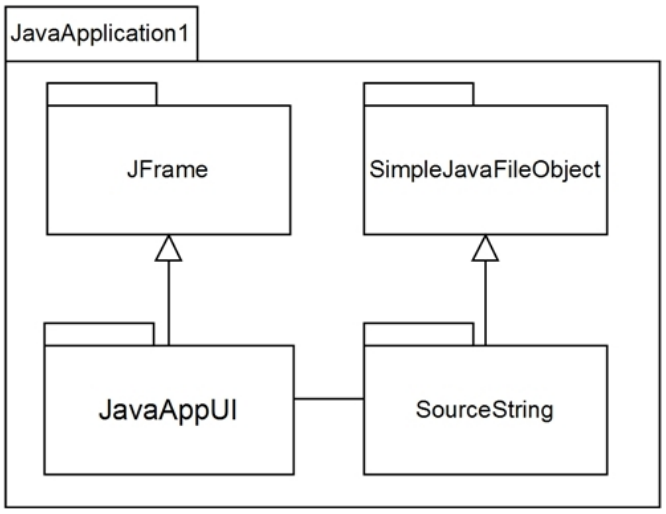
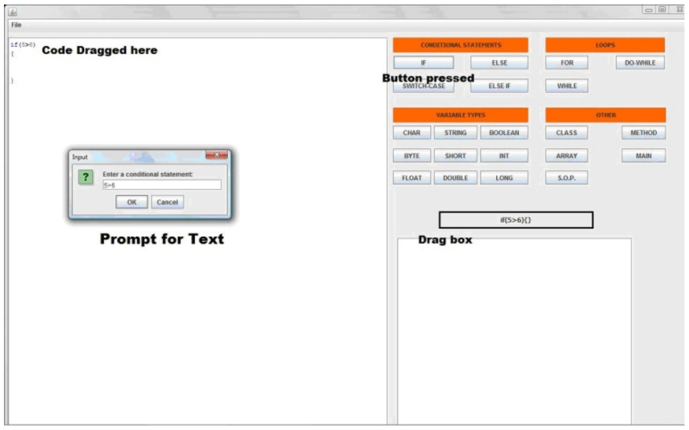


Fig. 1 the package diagram of Dragon Drop

A. Minimizing the Syntax Errors

The aim was to lessen the frustration experienced by novice programmers by decreasing the amount of syntax mistakes made while creating a program. To do this, we used buttons that prompt the student for information needed to create the code for them to remove the chance for syntax errors (Figure 2). We accomplished this based on the order the course materials are given to the students. The program has several available buttons for creating code that they would need the most.



```
if(5>6)
{

}
```

Enter a conditional statement: 5>6

if(5>6){}

Fig. 2 prompt for information

This process still teaches the students to code syntax; the code is not just created for the students. The students are shown the source code before they drag it to the text area. Once the student has entered the information required by each button, the generated code is sent to a text window under the buttons. The sequence diagram which describes the creation of the code, editing the source code and compilation is shown in Figure 3.

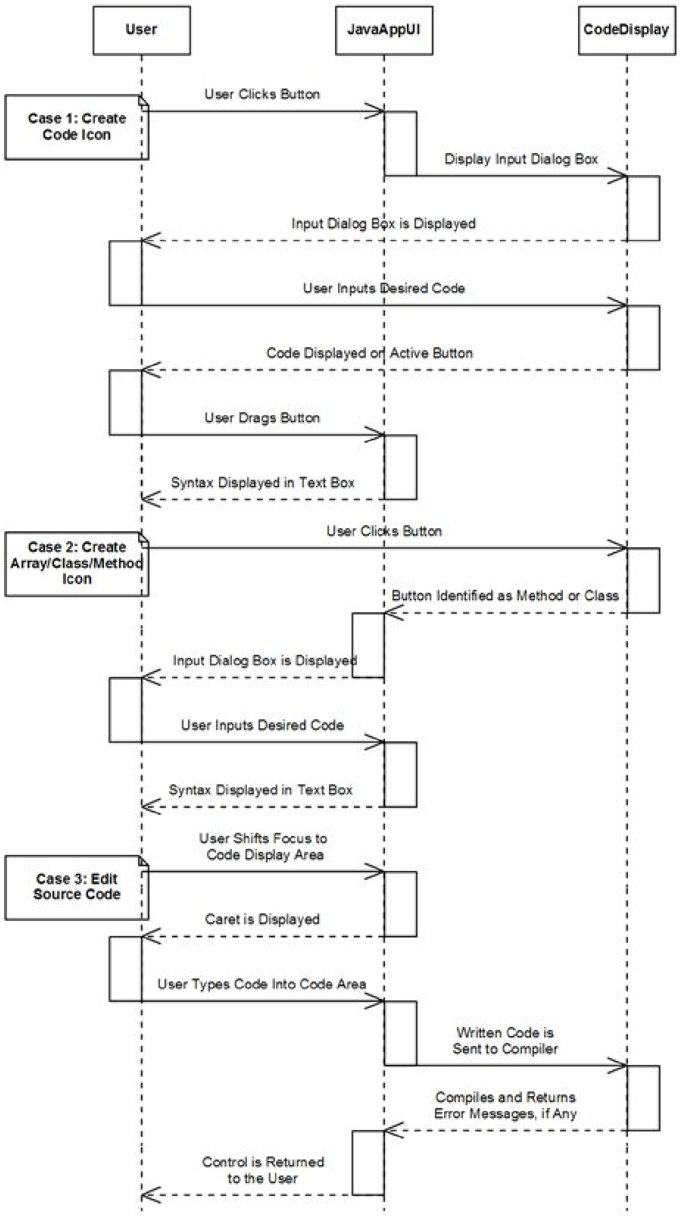

User
JavaAppUI
CodeDisplay
Case 1: Create Code Icon
User Clicks Button
Display Input Dialog Box
Input Dialog Box is Displayed
User Inputs Desired Code
Code Displayed on Active Button
User Drags Button
Syntax Displayed in Text Box
Case 2: Create Array/Class/Method Icon
User Clicks Button
Button Identified as Method or Class
Input Dialog Box is Displayed
User Inputs Desired Code
Syntax Displayed in Text Box
Case 3: Edit Source Code
User Shifts Focus to Code Display Area
Caret is Displayed
User Types Code Into Code Area
Written Code is Sent to Compiler
Compiles and Returns Error Messages, if Any
Control is Returned to the User

Fig. 3 sequence diagram of the editor

B. Immediate Gains

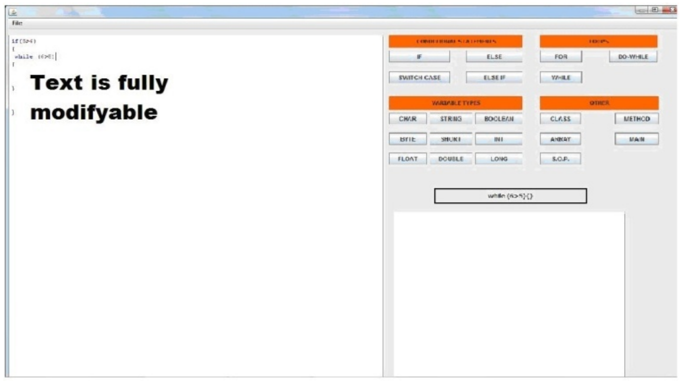
The next area of the program we addressed was showing students the immediate process. The student can click the button and after the input is entered the code is generated, and immediately the student can see what the code looks like without have to consult texts or references.

After the code is created for the students, they can drag the code directly to the text area and as they view the code. The student gains immediate gratification as the code is in the program and ready for use. The student can compile immediately and see how it affects the program.

C. Easy of Transition

The ease of transition is one of the most important parts because as the students' progress into next levels of programming they will need to be able to transition to other programming environments. We wanted to make sure that the students have modern conveniences of some of the other programming environments as well.

The text area is the same type of function that would be found in a modern programming environment as well as tabbing and the colour of the text for easy visual. The transition to another program will be easier based on familiarization of the environment. Although acting different with the drag-and-drop functionality, it will look similar to most other programming environments. The text after being dragged to the text area is fully useable and ready for editing (Figure 4).



```
if (5>6)
{
 while (6>5)
```

```
while (6>5){}
```

Fig. 4 fully editable text area

IV. Test Case and Results

A. Survey

The test cases used to determine how *Dragon Drop* helps students program and how they feel about programming was first a preliminary survey taken in the first quarter of the semester, and another taken closer to the end of the semester. In the first survey, the questions were focused on students' major, background, computer skills, the time spent on syntax problems vs. logical problems and their interest in computer science major. After which, we let the students use *Dragon Drop* closer to the end of the semester once they had been programming with a plain text editor. The testing phase of the program was having the Introduction to Programming students try out *Dragon Drop*. We replaced one of the labs the students were required to accomplish with another one that tested all phases of our application. Then we administered a post test survey to see how they felt about *Dragon Drop*, what they preferred and what helped them the most. Specifically, in the second survey, the questions more focused on their feelings on the *Dragon Drop* and their interest level in computer science. Both surveys were done in a 1 to 5 point format, 5 being very well liked and 1 being much disliked or little progress made.

The students who completed the surveys were from a variety of different majors and programming backgrounds. Many were beginner while some are intermediate programmers. All students were required to test *Dragon Drop* and give feedback. The details about the student distribution are given in Figure 5. The results on average were positive and the application was well received with few bugs.

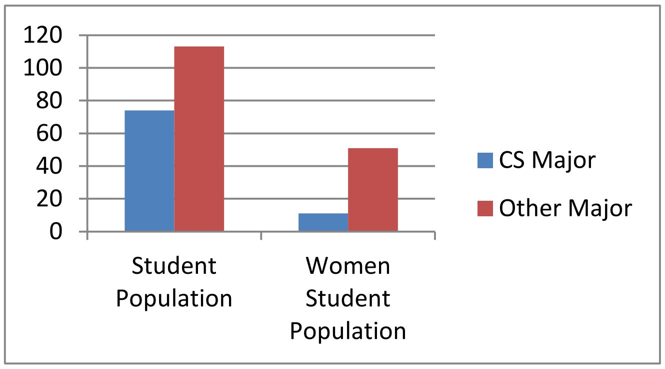


Fig. 5 participants

B. Student Diversity and Computer Programming Experience

Programming Fundamentals – I course is both a core course for Computer Science majored students but also an elective course for Math, Physics, Criminal Justice, and Computer Animation majored students. Therefore, more than half of the students participated in the survey were majored in a different area than Computer Science. As shown in Figure 5 above, almost 4 out of 5 women students who participated in the survey are not-CS majored students. Since almost all of the students are holding a high school degree from the state of Texas some of them had the opportunity to take Java programming course prior to entering to the college. Other than the education gathered from the school, because of their interest some of them also had programming skills while started to take the Programming Fundamentals – I course. The detailed student numbers about prior programming experience are shown in Figure 6.

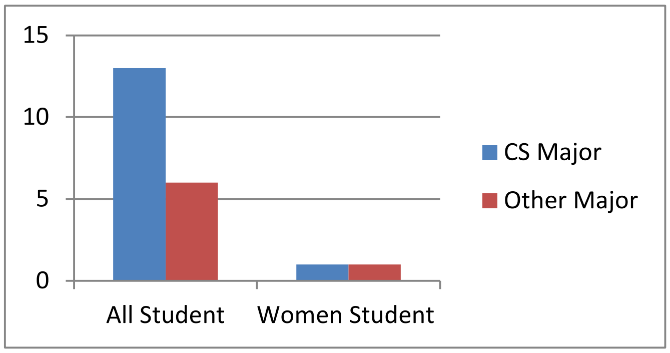


Fig. 6 prior programming experience

C. Main Challenges in Programming Course

After the students had initial experience with programming and conducting couple of labs, we wanted to see the main challenges that they face in this class and compare it to the survey that is conducted at the end of the semester. Since the students who had prior experience with programming languages are not faced vital challenges throughout the semester, their reflection is omitted. In the first survey, as shown in Figure 7, the students who don't have any prior programming experience claimed that the main challenge is the time spent on fixing syntax errors, followed by defining variables, and then the logical issues, true for all CS major, other major, and non-CS majored women students. However, as shown in Figure 8, throughout the semester the main problem shifted to logical problems in favour of 72 percent for the non-CS majored women students.

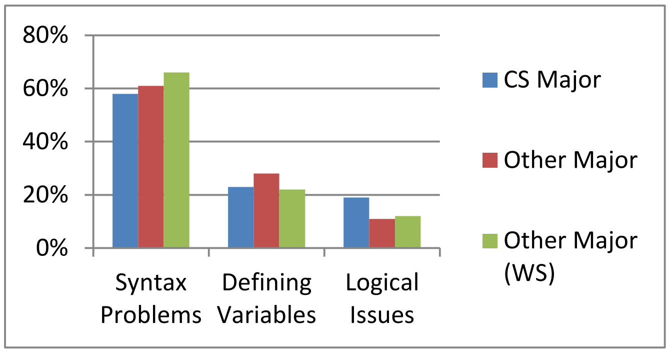


Fig. 7 main challenges at the beginning of the course

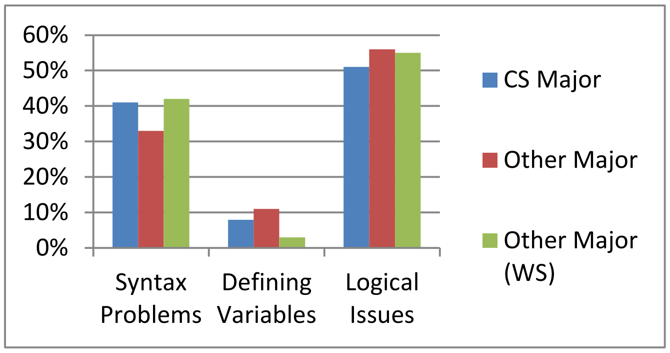


Fig. 8 main challenges at the end of the course

D. About the Dragon Drop and Student Interest in Computer Science

Another question asked between the two surveys was how the students felt about Computer Science in general. As reflected in Figure 9, in the first survey, only 4 percent of the non-CS majored women students had a very high interest in Computer Science. The students were frustrated at the class and the field in general about programming. After the second survey, 38 percent of the non-CS majored women students felt very good about the field of Computer Science. 88 percent of the non-CS majored women students agreed that the *Dragon Drop* application will address the syntax problem and decrease the time spend in debugging and will yield to an increase in the interest in Computer Science major.

As shown in Figure 10, 68 percent of the non-CS majored women students claimed that this application will be useful for throughout the course, from the first java program to array concepts, including primitive data types, conditional statements, loops, methods and objects. On the other hand, 12 percent of the non-CS majored women students do not see a need for the application. The remaining 20 percent of the non-CS majored women students indicated the usefulness of the tool for the first weeks of the semester, which includes the topics from first java program to loop structures. Those students claim that after the first initial weeks, the students' will have fewer errors to fix in their coding. When we look at final grades obtained from the course, the non-CS majored women students who provided the opinion to use the *Dragon Drop* throughout the semester had an average of 74, the women students who like to see the tool to be used in the first part of the semester had an average of 81, and the rest of the students who doesn't see a need for the tool to be used in the class had an average grade of 87 as shown in Figure 11.

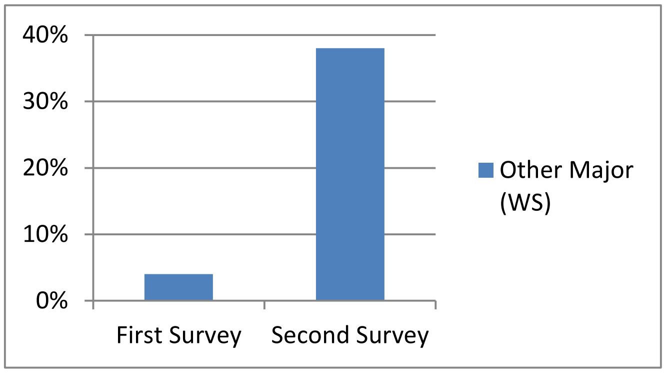


Fig. 9 interest in computer science among non-CS majored women students

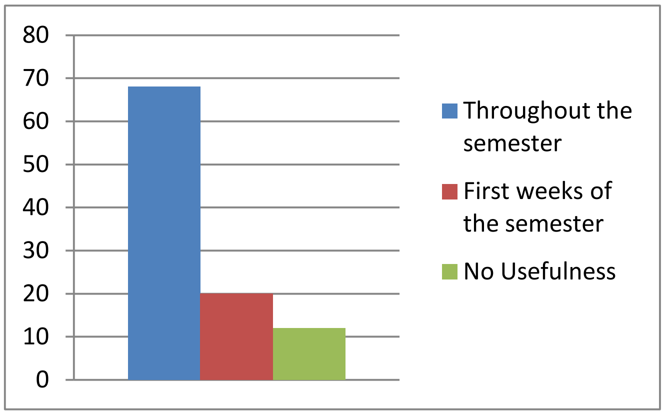


Fig. 10 usefulness of dragon drop among non-CS majored women students

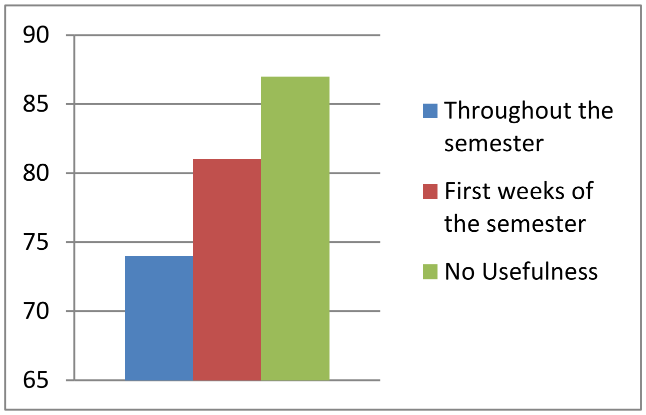


Fig. 11 final course grade average for non-CS majored women students

The Figure 11 proves the fact the application is correctly addressed for the women students who struggled more in this class. Moreover, in a separate question, 86 percent of the students who do not have any prior programming experience claimed that this application would be much more useful at the beginning of the semester compared to the end of the semester. Also, 82 percent of the non-Computer science majored women students prefer to use this application instead of current programming environment.

V. Conclusions

Looking at the survey data by major, the highest marks for interest in using *Dragon Drop* for the whole semester are from the non-CS majors and especially from women students. The fact that highest marks came from all non-CS majors suggests one of the major goals of this program was a success. If the non-majors prefer *Dragon Drop* over *TextPad* [20], they may have felt it lessened their frustration and increased their interest in Computer Science. What possibly might have contributed to this was students seeing what could be accomplished by their fellow students in *Java*, if they take the time to learn.

With this application, it is evident that, non-CS majored women students' level of frustration will be decreased and interest will be increased due to the ease of programming with the application. However, when we evaluate whether the students would prefer to use *Dragon Drop* over *TextPad* for the whole semester, only about 12 percent of the CS majored students preferred to use *Dragon Drop* over *TextPad*. This number is relatively low considering those students have seen other programming environments. Knowing this, it slowed them down as they can type code much faster than buttons allow. Moreover, the full statements generated by the application may negatively affect the CS majored students. For instance, since the students will have to take other upper level programming courses, the more they need to debug at the beginning will yield to less time spend in fixing common syntax problems in the following programming courses. Also, for a software developer position, the market will be essentially recruit recent graduates who have not only good code writing skills, but also a good debugger as well. In the long run, this tool may hurt CS-majored students' job search. But also they claimed that the moving from this editor to another editor won't be a challenge because of the simpler user interface and coloring schemes used in the application.

Overall, although the response for the initial survey was satisfactory, still there are several issues that need to be evaluated. As a future work, the application will be employed as the core text editor for the non-CS majored students in Programming Fundamentals I class. Learning outcomes will be evaluated based on the women students' performance, namely test scores, lab work, total debugging time. Also, we will start evaluating if having a *Dragon Drop* type of programming environment can attract non-CS majored freshmen or sophomore women students to change their major or include a minor in Computer Science field.

Acknowledgment

The authors are to extend recognition to Olamide Kolawole, David Key, and Brad Houck for being helpful during the implementation stage of the *Dragon Drop*.

References

[1] S. Zweben, "Computing Degree and Enrollment Trends", 2010-2011 CRA Taulbee Survey, *Computing Research Association*. 2012. Available: http://www.cra.org/govaffairs/blog/wp-content/uploads/2012/04/CS_Degree_and_Enrollment_Trends_2010-11.pdf

[2] S. Garner, "The Cloze Procedure and the Learning of Programming", *8th WSEAS International Conference on COMPUTERS*, Athens, Greece, 2004

[3] S. Dehnadi and R. Bornat, "The camel has two humps" (working title) [online], Middlesex University, UK, 2006 Available: http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

[4] J. Roberts and R. Styron, "Student Satisfaction and Persistence: Factors Vital to student retention", *Research in Higher Education Journal*, volume 6: 1-18. 2011.

[5] J. M. Cohoon. "Toward improving female retention in the computer science major". *Communications of the ACM*, 2001.

[6] R. Hanzu-Pazara and Eugen B, "Teaching techniques –modern bridges between lecturers and students", *7th WSEAS International Conference on Engineering Education (EDUCATION '10)*, Corfu Island, Greece July 22-24, 2010, ISBN: 978-960-474-202-8

[7] D. T. D. Phuong, F. Harada, H. Takada, and H. Shimakawa, "Collaborative Learning Environment with Convincing Opinions for Novice Programmers", *5th WSEAS / IASME International Conference on ENGINEERING EDUCATION (EE'08)*, Heraklion, Greece, July 22-24, 2008

[8] J.A. Marin-Garcia, and J. L. Mauri, "Teamwork with University Engineering Students. Group Process Assessment Tool", *Proceedings of the 3rd WSEAS/IASME International Conference on Educational Technologies*, Arcachon, France, October 13-15, 2007, pp. 391 – 396

[9] J. A. Betancur, C. Rodríguez, and I. Ezparragoza, "An undergraduate collaborative design experience among institutions in the Americas", *Proceedings of the 8th WSEAS International Conference on Engineering Education (EDUCATION '11), Proceedings of the 2nd International Conference on Education and Educational Technologies 2011 (WORLD-EDU '11),* Corfu Island, Greece July 14-16, 2011, ISBN: 978-1-61804-021-3

[10] M. Blaho, M. Foltin, P. Fodrek, and J. Murgas, "Students perspective on improving programming courses", International Journal of Education and Information Technologies, Issue1, Volume 6, 2012

[11] C. McDowell, L. L. Werner, H. E. Bullock, and J. Fernald, "Pair programming improves student retention, confidence, and program quality*". Communications of ACM* 49(8): 90-95 (2006)

[12] W. Dann and S. Cooper, "Education Alice 3: Concrete to Abstract," *Communications of the ACM 52.8,* 2009.

[13] J. C. Cheung, G. Ngai, S. C. Chan, and W. W. Lau, "Filling the Gap in Programming Instruction: A Text-Enhanced Graphical Programming Environment for Junior High Students," *Technical Symposium on Computer Science Education*, 2009.

[14] P. Henriksen and M. Kolling, "Greenfoot: Combining Object Visualization with Interaction," *Conference on Object Oriented Programming Systems Languages and Applications*, 2004.

[15] S. Kouznetsova, "BlueJ and Blackjack to Teach Object-Oriented Design Concepts in CS1," *Journal of Computing Sciences in Colleges* 22.4, 2007.

[16] K. J. Goldman, "A Concepts-First Introduction to Computer Science," *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004.

[17] S. Fincher, S. Cooper, M. Kolling and J. Maloney, "Comparing Alice, Greenfoot & Scratch." *Technical Symposium on Computer Science Education*, 2010.

[18] T. Daly, "Using Introductory Programming Tools to Teach Programming Concepts: A Literature Review," *The Journal for Computing Teachers*, 2009.

[19] J. M. Lavonen, V. P. Meisalo, and M. Lattu, "Problem Solving with an Icon Oriented Programming Tool: A Case Study in Technology Education," *Journal of Technology Education, 2001*.

[20] Helios Software Solutions, *TextPad*. Longridge, England: Helios Software Solutions, 2010.

Shannon Silessi is a M.S. student in the Department of Computer Science at Sam Houston State University, Texas, USA. Her research interests are Software Engineering and Information Assurance.

She earned her B.S. degree from Computer Science at Sam Houston State University, Texas, USA in 2011.

Cihan Varol is an Assistant Professor of Computer Science at Sam Houston State University. His research interests are in the general area of information (data) quality, VoIP Forensics, and risk management with specific emphasis on personal identity recognition, record linkage, entity resolution, pattern matching techniques, natural language processing, multi-platform VoIP applications, VoIP artifacts data cleansing, and quality of service in business process automation. These studies have led to more than 30 peer-reviewed journal and conference publications, and one book chapter.

He received his Bachelor of Science degree in Computer Science from Firat University, Elazig, Turkey in 2002, Master of Science degree from Lane Department of Computer Science and Electrical Engineering from West Virginia University, Morgantown, WV, USA in 2005, and Doctor of Philosophy in Applied Computing from University of Arkansas at Little Rock, Little Rock, AR, USA in 2009.

Hacer Varol is an Instructor at the Department of Computer Science at Sam Houston State University. Her research interests are in the general area of biomedical signal processing, educational technology, and space network security.

She received her Bachelor of Science degree in Electrical and Electronics Engineering from Firat University, Elazig, Turkey in 2003, Master of Science degree from Applied Science department from University of Arkansas at Little Rock, Little Rock, AR, USA in 2011, and currently pursuing doctorate degree in Electrical Engineering from Lamar University, Beaumont, TX, USA.