Knowledge Building Using Visualizations

Vijayakumar Shanmugasundaram
Department of Math and CS
Concordia College
Moorhead, MN, USA
218-299-3343
shanmuga@cord.edu

Paul Juell
Department of CS and Operations Research
North Dakota State University
Fargo, ND, USA
701-231-8906
paul.juell@ndsu.edu

Curt Hill
Department of Mathematics
Valley City State University
Valley City
ND, USA
701 845-7103
curt.hill@vcsu.edu

ABSTRACT
In this paper, we describe our efforts in knowledge building by creating visualizations. Our efforts include problem-based learning. We have identified a problem that the students have in learning OOP. To solve that problem we engage our students in the classroom using existing visualizations created by students of the earlier class, then we have the currents students improve the existing visualizations, or create new visualizations for future use in the same class. We describe the process of building knowledge, problem based learning, the details of the visualizations, our observations and the merits of this approach.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education.*

General Terms
Design, Experimentation, Human Factors, Languages.

Keywords
CS educational research, knowledge, visualizations, OOP.

1. INTRODUCTION
Knowledge building is a process of producing and continually improving valuable ideas to a community, through ways that raise the possibility of community accomplishment being greater than the total individual contributions and part of broader cultural efforts [7]. Many organizations participate in progressive knowledge building [2]. Educational organizations show interest in promoting knowledge building among learners. However, knowledge building can be difficult to introduce in classrooms, especially in an introductory programming class. The main reason is that courses often engage students in contrived programming assignments that are mainly as exercise language features. These artificial assignments provide no value in terms of communal knowledge advancement.

ITiCSE'06, June 26–28, 2006, Bologna, Italy.

We have students in an introductory programming language class actively engage in knowledge building by teaching them with the knowledge (visualizations) built by a previous class, by giving problem based learning programming assignments which will help them to refine the already built knowledge (visualizations) and to build new knowledge (new visualizations) for future use in the same class [5].

2. BUILDING KNOWLEDGE PROCESS
Teaching introductory object oriented programming with the help of conceptual diagrams, UML diagrams [1], and memory diagrams motivate and help students to learn object oriented programming [6, 8]. We initially created visualizations to help the students learn OOP. For the last three years these have been used in teaching an introductory programming class in Java at Concordia College. This introductory programming class is one credit course out of 32 credits required to graduate with classes meeting every week MWF for 1 hr 10 minutes. The building knowledge process includes the instructor initially creating visualizations and using them in the classroom, then requiring students to create visualizations as final projects for this introductory programming class. We also allow them to refine the existing visualizations for use in future classes.

Throughout the semester, the concepts are taught with the help of diagrams. In the beginning of the semester the students are informed that they need to choose a concept in the course and create visualizations for their concepts in teams of two. We steadily prepare them towards that end by continually teaching them with the aid of visualizations, and having the students create diagrams (UML diagrams or memory diagrams) in a word-processor, and then display them on their web pages as part of their assignments. Initially they do not have enough background to create a diagram of their own in Java, but after applets are covered they do. Then the students create a class diagram, using applets and display this diagram on their web pages. Just one month before the end of the semester, they are given a programming assignment in event handling and again have them draw a memory diagram using an applet. This applet will have text fields, buttons to collect the name of the class, object variable, instance field. After collecting the data from the user, clicking on the button will show a memory diagram. Again the students need to display this applet on their web pages. This particular assignment becomes a good starting point for their final visualization project. The students have the option to choose the type of diagram that they want to create from the following: Class diagram, Memory diagram, or Conceptual diagram. They may also choose to improve an existing visualization with better user

interfaces. The students need to complete this final project in only two class periods. However, they have been prepared to this end from the beginning. This final project is 20 % of the grade.

During this time, we have been building and refining knowledge and improving the instruction of the course. The students have built enough visualizations for each concept. Therefore we only use student created visualizations in our classroom. Next will be discussed some of the interesting visualizations that the students have made, refined, and used as well as the importance of these visualizations as teaching tools, and why and how they are used in the classroom.

3. DETAILS OF VISUALIZATIONS

3.1 Interfaces

An interface is a collection of operations that specifies the services of a class or component. It describes the externally visible behavior of that element. It defines a set of operation specifications (that is, their signatures) but never a set of operation implementations. It rarely stands alone, rather, it is typically attached to the class or component that realizes the interface. Graphically, an interface is rendered as a circle. An interface is also rendered as a class diagram with a slight change in the notation. Consider the UML interface diagram shown in Figure 1.

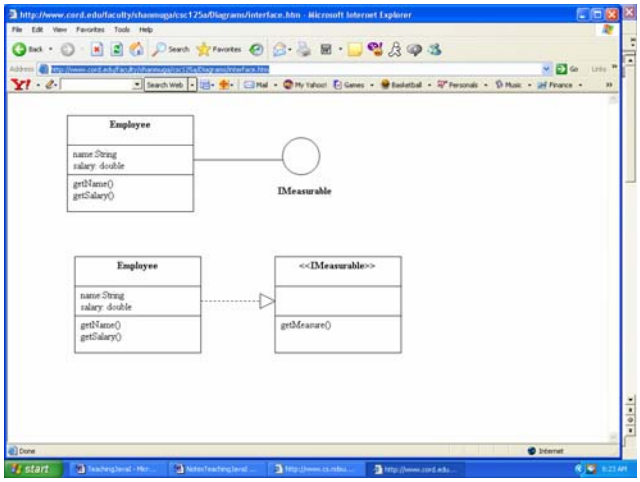

Figure1. UML Interface diagram

Employee is a class that realizes the IMeasurable interface, which is rendered as a circle. In the alternate representation, the interface is represented as a class diagram with a slight change in the notation - the name of the interface is within angle brackets. This IMeasurable interface has a getMeasure method. The relationship between class and the interface is represented by the realization symbol. This UML representation shows the external view, internal view of the interface, and its relationship to a class, which realizes it. These diagrams are used to teach the concept of an interface to the students.

The students identified that this UML diagram does not show why an interface is needed in the first place, and suggested that in order to explain the necessity of an interface the conceptual diagram is required. Interfaces reduce the coupling between classes, promote reuse, and ease the changes being made with the classes that realize an interface and the classes that depend on the interface. Consider the following conceptual diagram involving an SUV, a trailer, a boat, and snowmobile drawn by a team of two students as shown in Figure 2.

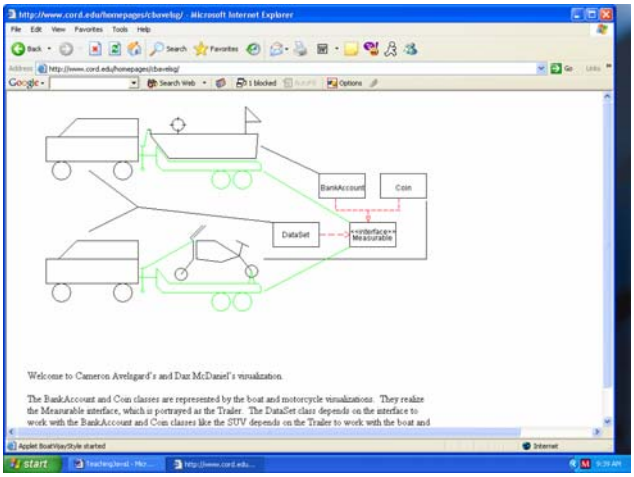

Figure 2. Conceptual diagram to teach an interface

The SUV depends on the trailer to carry either the boat or the snow mobile (dependent relationship). Both the boat and snow mobile have provisions for being placed and fixed on the trailer (Realization relationship). If the trailer is not there, every time the SUV needs to tow either a boat or snowmobile, it has to change to an appropriate coupling. But with the help of trailer, without changing its coupling, it can carry either the boat and the snow mobile as long as the boat and snow mobile has the provision to be fixed on the trailer. More over any alterations can be easily made to SUV, boat, snow mobile, and the trailer as long as the relationships are maintained. An interface cannot stand alone like this trailer, which cannot travel on its own. With the help of this diagram, the students clearly explained the concept of an interface. The mapping of the classes involved and the interface with the SUV, boat, snow mobile, and trailer enhances the understanding of the concept of the interface.

3.2 Polymorphism

Polymorphism is the ability for different classes of objects to interpret the same message differently. Objects are responsible for interpreting the messages sent to them. Dynamic binding facilitates polymorphism. This allows the developer to let the system decide which method should be called on what object, thereby avoiding the writing of complicated decision logic. The facility that makes a developer's job easier, makes the student's job of understanding polymorphism more difficult. We taught polymorphism by creating an UML class diagram showing the relationship between two classes, one interface, and another class given in the textbook [3] we used as shown in figure 3. We tried to explain how a method specification provided in the interface is implemented in the first two classes that realize the interface differently. Testing the class that uses the third class is complicated by the dependence on the interface for a method with same name and signature. This same method acts differently

depending on the objects of the two classes that realize the interface. Explaining this situation to students in just words can be quite vexing. It is difficult to provide a conceptual diagram for a BankAccount class, Coin class, Mesurable interface and DataSet class to explain Polymorphism. There this visualization shown in Figure 3 is used as well as for explaining upcasting and downcasting.

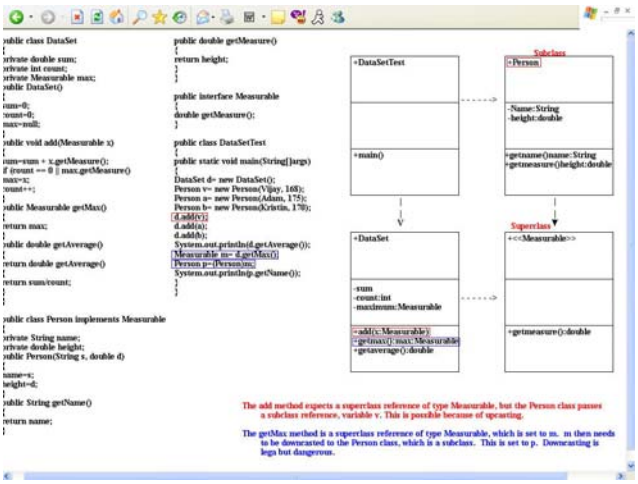

Figure 3. A class diagram to teach polymorphism

This becomes easier to understand when given a simple example that relies upon their common knowledge. The students suggestion is to simulate a program for different animals shouting. Sending the "shout" message to a cow object, a pig object, or a duck object produces different results although the message is the same. This visualization is shown in figure 4. The students who created this visualization combined all levels of knowledge at the conceptual level, code level, and memory level. Initially they were interested only in creating the visualization at the conceptual level, but it was suggested that to connect it at all levels would greatly increase its value. The students then created the visualization combining all levels of knowledge.

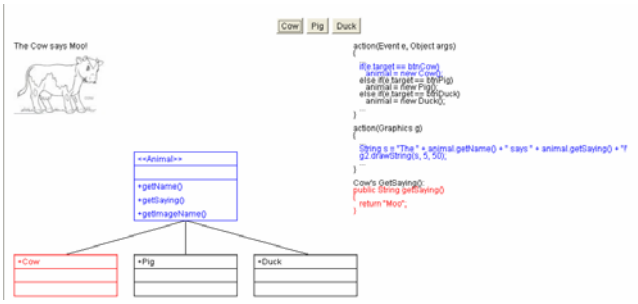

Figure 4. Conceptual diagram to teach polymorphism

3.3 Operator Overloading

Another problematic concept is function name and operator overloading. This is often combined with an explanation of the difference between early and late binding. This has been done without any diagrams. Our students created a visualization to explain operator overloading by combining the explanation at all levels as shown in the figure 5.

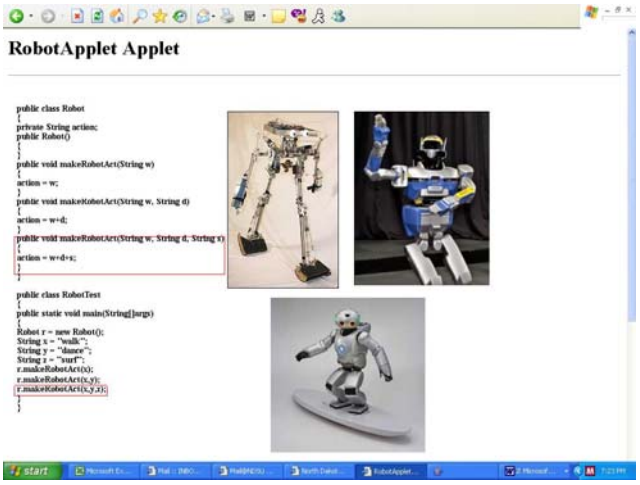

Figure 5. A conceptual diagram to teach operator overloading.

The students created three different methods with the same name "makeRobotAct" with three different parameters. First method to make the robot walk, second method to make the robot walk and dance, third method to make the robot walk, dance, and surf. While this visualization clearly explains the method overloading/operator overloading, there is still a need to come up with another visualization to explain the difference between early binding and late binding.

There are many other equally useful visualizations that must be omitted for want of space. The interested reader should consult the web site [9]. We want to highlight some interesting observations we made during the time when the students were creating visualizations.

4. OUR OBSERVATIONS

The students showed more interest in developing a conceptual diagram without making a connection to the lines of code as shown in Figure 6.

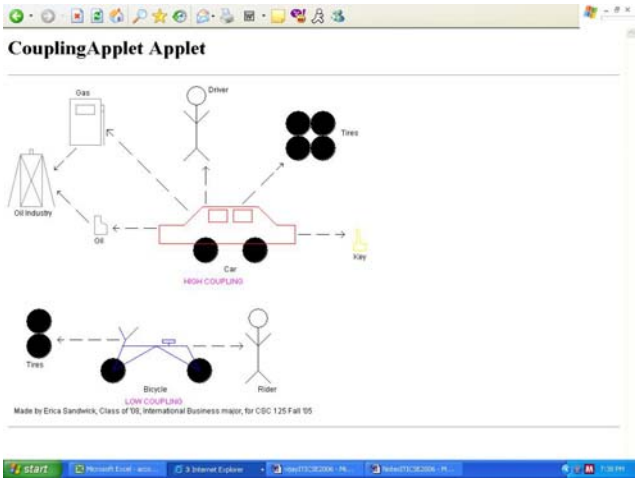

Figure 6. A conceptual diagram to teach coupling

They need to be encouraged to create visualizations by making a connection between the diagrams and the lines of code. We pointed out the students' difficulties in converting the concepts to code as the main reason that they need to make the connection between the lines of code and the diagrams in the visualizations.

They also try to put more emphases on their visualization as shown in Figure 7 especially when it comes to a particular point where it is more abstract such as listener in Event Handling. The students don't have difficulty in understanding the code that creates a window frame or a button being added to the window. But the students have difficulty in understanding the workings of a listener, which can be seen by the ear listening picture in the visualization.

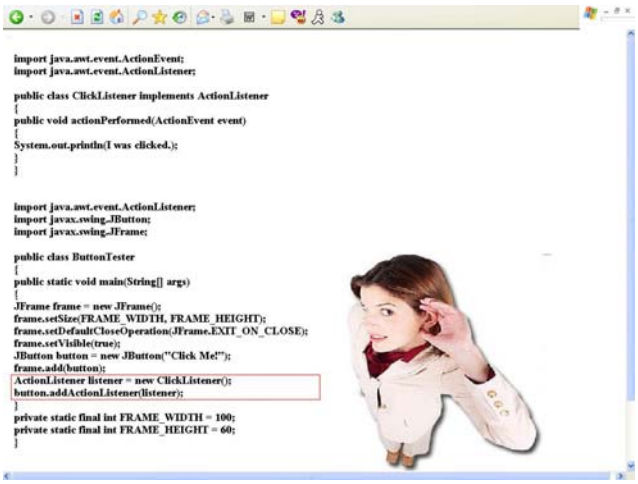

Figure 7. A conceptual diagram to teach event handling

It is interesting to note that the students always add some element of humor in their visualizations as shown in figure 8.

Figure 8. A conceptual diagram to teach while loop

It is important to note that the students try to identify the important concepts to something concrete in the real world. For example, to explain a while loop with a guarded command, the students identify the condition in the while loop to a lock. The visualization shows as long as the cage door is locked, the monkey will be inside. Once the door is unlocked, the monkey will be released. This visualization clearly explains the guarded commands with a concrete example.

The first author also routinely uses the visualization as a reference in his recommendation letter to future employer of the students. These visualizations are the living example of the student's capacity in the course. The students are passionate about creating the visualization. This is mainly because that they think that they are trying to solve their own problems. For example, in one class, the discussion went about trying to compare the iterative implementation and recursive implementation of the same problem. The instructor attempted to explain this, but one of the students who was looking for a concept to visualize was initially unable to understand the difference between these two implementations. At the end of the class discussion, the concerned student took this concept as a theme for his visualization.

It is very unfortunate that the students creating these visualizations do not realize the importance of providing narrative comments as par of the visualizations. Narrative comments included in the visualizations can improve the efficiency of visualization. While many students after our suggestion understood the importance of narrative comments and wanted to implement it. This forced us to explain to them about threads and concurrency control in the programs to provide the appropriate narrative comments at the right time. Only a few students ventured on their own to use threads in the visualization for including the appropriate narrative comments at the right time as show in figure 9. This visualization explains the execution of a program with exception handling code in it using appropriate running commentary about the program being executed.

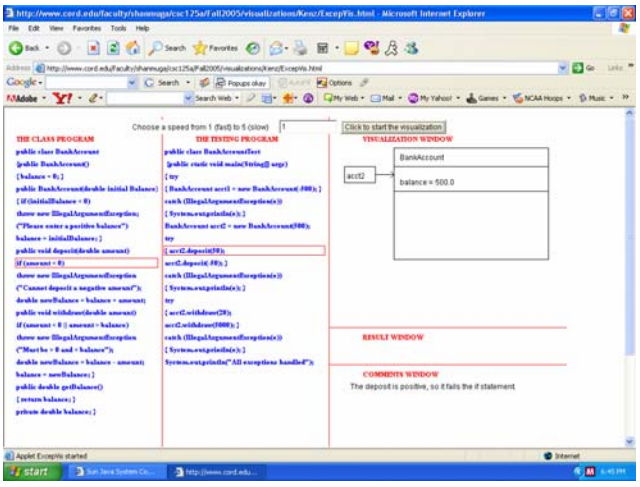

Figure 9. A diagram with narrative comments

Most of the students want to create their own visualization from scratch. For some reason they hesitate to select the visualization created by another student in the earlier class and make

improvements on it. Students need to be encouraged to take an existing visualization and make improvements, since this will be a useful skill when need to modify existing software.

Another interesting observation is that the students were resistant when they start their visualization project, but once they started, then it was very hard to stop them as they get more ideas and want to implement all these ideas.

In one of the parent/faculty meeting, one parent of the student had approached the first author and said that he is proud of the visualization his son created for this class. This parent printed and made a frame of the visualization and kept it next to the drawing his son made in his kindergarten class.

5. MERITS OF THIS APPROACH

Implementing knowledge building in an introductory programming course is difficult. However, with careful planning and student preparation, it becomes possible and beneficial. The typical contrived assignments are not helpful. Instead they concentrate on making the students to go through the process of writing the code and not on the process of making them understand the concept. The aim for the students with the typical assignments is to get a grade, whereas this type of project promotes the understanding of the concept. The majority of the made up assignments are graded, handed back to the students, and forgotten before the end of the semester. These visualization projects tend to have a longer lifespan. They remain on the web for the next set of students [9] and are used by students and instructors from other institutions as well. Our visualizations indicate what the students do not fully understood about a concept. The retention of the visualizations discourages cheating attempts by the students, as they know that their final project is going to be on display for some time.

6. FUTURE WORK

We intend to determine the significance of this approach by conducting a more rigorous experiment with an experimental and control group. We will continue to engage our students in future classes to refine the existing visualizations.

We have identified a number of problems associated with teaching the CS Introductory programming course in Java. The students do not have enough tools to master the concepts by practicing them. For example, we may require the students to do one or two assignments to create a class program, or to write a recursive method. But doing these assignments once does not reinforce the process of creating a class program or recursive method. Hence, we are planning to have students build expert systems. These expert systems would guide the students with various steps involved in creating a class program, or a recursive method. These also would be used by the future students to practice and master the process of creating similar programs. Moreover, the Java Task Force [4] identified a number of problems associated with teaching introductory programming course in Java. We plan to have our students in the future class to solve them one by one and build a knowledge system that can help the students to properly learn the programming concepts and practice.

7. CONCLUSIONS

We identified the problem that students have in learning OOP. To solve this problem we use visualizations to teach the concepts. Then we engage our students to use existing visualizations created by students of the earlier class, and have them improve the existing visualizations, or create new visualizations for future use in the same class.

8. ACKNOWLEDGMENTS

Our thanks go out to many students of Concordia College who have contributed to this project.

9. REFERENCES

[1] Booch, G., Rumbaugh, J., and Jacobson, I. *The Unified Modeling Language User Guide.* Addison Weisely. 1999.

[2] Hewitt, J. *From a focus on tasks to a focus on understanding: The cultural transformation of a Toronto classroom.* In T. Koschmann, R. Hall, & N. Miyake (Eds.) Computer Supported Cooperative Learning Volume 2: Carrying forward the conversation (pp. 11-41). Mahwah, New Jersey: Lawrence Erlbaum Associates.

[3] Horstmann, Cay. *Java Concepts*, 4th ed. John Wiley, New York, NY, 2006.

[4] Java Task Force Report. http://www-cs-faculty.stanford.edu/%7Eeroberts//java/java-problem-taxonomy.html, Date accessed on 12-20-2005.

[5] Juell, P., & Shanmugasundaram, V. *Learning Object Oriented Programming By Creating Visualizations.* The 19th International Conference on Computers and Their Applications (Sponsored by the International Society for Computers and Their Applications (ISCA)), Red Lion Hotel on Fifth Avenue, Seattle, Washington USA. March 18-20, 2004.

[6] Juell, P., Shanmugasundaram, V., & Denton, A. *Effectiveness of Visualizations for Student Use.* IED-MEDIA 2003-World Conference on Educational Multimedia, Hypermedia & Telecommunications, Association for the Advancement of Computing in Education (AACE). Honolulu, Hawaii, USA.

[7] Scardamalia, M., & Bereiter, C. *Knowledge building.* In Encyclopedia of Education (2nd ed., pp. 1370-1373). New York: Macmillan Reference, USA.

[8] Shanmugasundaram, V., Juell, P., Jayasuriya, N., and Benson, J. *Development of ViewJ - a Visualization Builder for Object Oriented Programming Development* Environment, IED-MEDIA 2004-World Conference on Educational Multimedia, Hypermedia & Telecommunications, Association for the Advancement of Computing in Education (AACE), Lugano, Switzerland, June 21-26, 2004.

[9] Visualizations web site. http://www.cord.edu/faculty/shanmuga/ Date accessed 12-20-2005.