From Objects-First to Design-First with Multimedia and Intelligent Tutoring

Sally H. Moritz, Fang Wei, Shahida M. Parvez and Glenn D. Blank
Computer Science and Engineering Department
Lehigh University
001-610-758-4605, 001-610-758-4867
{sgh2,faw2,smp9,gdb0}@lehigh.edu

ABSTRACT
"Objects-first" is an increasingly popular strategy for teaching object-oriented programming by introducing the concepts of objects, classes, and instances before procedural elements of a programming language. Still, this approach emphasizes coding rather than other critical aspects of software development, notably problem-solving and design. We propose a "design-first" curriculum, which subsumes an objects-first approach into lessons that also introduce object-oriented analysis and design, using elements of UML before implementing any code. We also present CIMEL ITS, an intelligent tutoring system that uses the design-first approach to help students of various learning styles in a CS1 course. It interfaces with an IDE we have chosen specifically to support the design-first curriculum, and CIMEL, multimedia courseware which has been shown to be effective in helping students learn object-oriented programming concepts.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: (*computer science education, curriculum.*)

General Terms
Design, Languages, Experimentation.

Keywords
Object-oriented programming, UML, Java, Eclipse, objects-first, CS1, Java in high schools, CIMEL, multimedia, courseware, IDE, intelligent tutoring system, student model, pedagogy, e-learning.

1. INTRODUCTION
Teaching object-oriented design and programming concepts is challenging. Research by Ratcliffe[8] has shown that lack of comprehension expressed by first year computer science students is a rising concern in academia. Significant numbers of students have difficulties in understanding the basics of object concepts and programming principles. McCracken[14] performed a study that suggested that in UK and USA, approximately 30% of students do not understand the basics. Clearly, new techniques and tools are needed to help students of all learning styles master the crucial concepts of object-oriented software development.

ITiCSE 2005, June 27 - 29, 2005, Universidade Nova de Lisboa, Monte da Caparica, Portugal.

Our research initially addresses this problem through the curriculum. A heuristic for curriculum development is: students learn best what they learn first. So we considered using an "objects-first" approach facilitated by BlueJ[12] and a BlueJ-oriented textbook[5]. Experimental results (described below) support an "objects-first" approach to teaching Java over a more traditional syntax-first focus, especially with the help with multimedia-guided instruction and exercises.

We noticed, however, that an objects-first approach by itself does not necessarily address other critical aspects of software development - how to analyze a problem. Bailie[4] found this issue is the greatest obstacle for students in CS1. The BlueJ environment hints at object-oriented design by presenting beginners with a simple UML-like design with which they interact before writing any code. But BlueJ does not reveal where these designs come from or invite students to create their own. Thus BlueJ, like earlier syntax-oriented approaches, still reinforces a misconception that software development is little more than coding. In an era of popular concerns about out-sourcing programming jobs, as well as ongoing difficulties attracting women and minorities into computer science, we believe it is a mistake to continue reinforcing this stereotype in the first course. Before coding, we believe students should get a glimpse of the big picture of software development, so that they can better appreciate its challenges and opportunities. We therefore propose a "design first" approach in introductory courses, in which students learn object-oriented analysis and design as problem-solving skills. In our curriculum, students learn how to develop use cases to analyze problems, then design solutions using UML, before implementing any code in Java. We have incorporated our new curriculum in first year computer science courses at Lehigh University as well as in a high school course at a local inner-city high school, and the results are promising.

An objects-first approach presents beginners with its own conceptual challenges. How much more so does a design-first approach. From design through programming, many students can benefit from one-on-one tutoring. Reiser[19] reported that students working with private tutors can learn given material four times faster than students who attend traditional classroom lectures, study textbooks and work on homework alone. Bloom[8] also reported that students have a better grasp of material working with a private tutor than attending traditional classroom lectures. Since it is not feasible that each student would work with a private human tutor, the next best option is an intelligent tutoring system. Anderson[3] reported that an ITS is a cost-effective means of one-on-one tutoring that providing novices with the individualized attention needed to overcome learning difficulties. Intelligent tutoring systems are not only being used in academia to augment classroom teaching but have also penetrated various industries where companies are using ITSs to train employees to perform their job functions. As a result, ITSs have been built for various domains such as medicine, engineering, public services, computer science, etc. The application of ITS in computer science has been limited to tutoring database design and specific procedural aspects of programming languages such as Java, C++, LISP and Pascal, and have not kept up with the current technology focus of object-oriented design and programming[17][15][13][19][20]. We present CIMEL ITS, an intelligent tutoring system that provides one-on-one tutoring to help beginners with various learning styles in a CS1 course learn the design-first approach.

CIMEL ITS will interact with students through two systems. One way is through an integrated development environment (IDE) in which students design and code their project solutions. Here, CIMEL ITS will observe the student's progress and offer assistance based on pedagogical strategies adapted to the individual. Another way is through multimedia courseware called CIMEL (Collaborative, constructive, Inquiry-based Multimedia E-Learning). CIMEL covers a breadth of topics in computer science, including introductory concepts in Java and object-oriented programming. The ITS can refer students to portions of CIMEL for review of topics needing reinforcement, and determine the student's level of understanding from the student's performance on quizzes and interactive exercises within CIMEL.

The rest of this paper is organized as follows: section 2 describes our design-first curriculum and how it improves on earlier approaches; section 3 discusses how multimedia facilitates objects-first learning; section 4 describes the structure of CIMEL ITS and how it will benefit the learner; and section 5 summarizes the project's current status and planned future work.

2. A DESIGN-FIRST CURRICULUM

2.1 Course Outline

The course (details about lesson plans are available at www.lehigh.edu/~sgh2) begins with a high-level introduction to software engineering in which the process of building an application is compared to building physical structures such as a building, bridge, or car. While exploring the analogy between building a house and software development, students learn about the steps of gathering requirements, creating a prototype or mock-up, figuring out a cost estimate, and creating documentation of all requirements. The planning and design phases are emphasized, with coding presented as the translation of a good detail design.

Next, students learn about objects, classes, and instances, first conceptually and then more concretely within Java. They complete a series of exercises in which they interactively create instances and manipulate them by invoking methods, using a set of classes that draw different shapes on a canvas. These initial exercises were adapted from the "Shapes" project used in [5].

Once students understand what a class is and how to use it, they tackle the problem of designing and building a small application from scratch. We introduce the design process via a sample project: building a vending machine to sell movie tickets. We emphasize the importance of first understanding the user's requirements and documenting them at a detailed level. The students identify the actors who interact with the ticket vending machine, identify use cases and draw a use case diagram, then document the steps, including alternatives, for each use case. Only after completing the use cases are the students allowed to proceed. At that point, they design the objects needed, define the attributes and behaviors of each, and create a class diagram. Students enter their class diagrams in the integrated development environment (described below), which generates the Java code for the class, attribute, and method definitions.

Next, students are introduced to basic procedural coding concepts in the Java language, including variables, primitive data types and Strings, assignment, if statements, and loops. As each concept is presented, students immediately employ it in coding one or more methods for the ticket machine. As the final phase of the project, students learn how to create a simple character-based interface for their system.

The concepts learned over the course of the first project are reinforced with a second project that the students complete on their own, with minimal assistance from their instructor. Finally, learn how to apply the Java Swing classes by building a graphical interface to the original ticket machine project, after which the students create a graphical interface for their own projects.

2.2 Integrated Development Environment

Using an integrated development environment (IDE) that supports the focus of the curriculum is critical to students' success. We needed an environment that met these requirements: 1. It allows entry of basic UML components (at minimum class diagrams). 2. It generates some basic code structures so students are shielded from some of the complexities of syntax. 3. It provides some interactive capabilities, such as allowing entry and execution of a single line of code. 4. It is not just a learning environment, but also a practical IDE that can help students make the transition to real world tools as they progress to more advanced courses. No

one IDE we considered had all these features, but we were able to put one together based on Eclipse.

Eclipse is an open-source IDE supported by a consortium of companies in the computer industry, including IBM and Sun, and widely used by developers in both academia and industry. Eclipse can be easily extended through the creation of "plug-ins," and many such extensions are available for a variety of purposes. For UML design, we chose the EclipseUML plug-in by Omondo.

DrJava is a simplified IDE designed for beginners which was developed at Rice University[2]. A key feature of DrJava is an "Interactions Pane" in which students can enter Java statements one at a time and see them execute instantly. We find this feature especially helpful in letting students practice creating instances and calling methods, because they can immediately see the results of each action. Fortunately, a version of DrJava is available as an Eclipse plug-in, and we included it in our course's IDE. Our default Eclipse setup (saving students the time to install the two plug-ins) is available at www.lehigh.edu/~sgh2.

3. CIMEL MULTIMEDIA AND EXPERIMENTAL RESULTS

CIMEL multimedia introduces the breadth of computer science and software development (a web-based demo and documentation is available at www.cse.lehigh.edu/~cimel), complementing a textbook, *The Universal Computer: Introducing Computer Science with Multimedia* [Blank 2003]. With multiple personae explaining and guiding learners in both audio and text boxes (audio only by default), JUST THE FACTS summaries, interactive, constructive and inquiry-based learning exercises, CIMEL facilitates learning for students with different learning styles [7].

In the CIMEL framework, we designed and evaluated multimedia paralleling the first chapter of the Barnes and Kölling textbook [5]. Seventy-six students in CS1 classes at Lehigh University participated. Half of them (randomly selected for the first time from all students) took a web-based pre-test of 20 questions. Half (randomly selected for the second time from all students) then used the text and BlueJ, while the other half used multimedia and BlueJ. All the students took a post-test. Later, the group that used the text and BlueJ was given access to the multimedia, and the group that used multimedia and BlueJ was assigned to read the book. A second post-test was given to all students. Thus the design of this experiment, based on the Solomon Four-Group Design, was—

R	O_1	X	O_2		O_3
R	O_1		O_2	X	O_3
R		X	O_2		O_3
R			O_2	X	O_3

—where R indicates random assignment, X indicates exposure to the multimedia tool, O_1 is the pre-test, and O_2 and O_3 are the two post-tests. This design checked for a testing effect, i.e., whether scores on the post-test are improved merely through practice on the pre-test. It also examined differences between the multimedia and text groups on the first post-test and a second time on the second post-test.

Taking the pre-test had no effect on the scores of the post-test. For the group getting the pre-test, the mean post-test was 13.44, while the mean for the group not exposed to the pre-test got a mean of 13.46, an insignificant difference ($t = .043$. $df = 72.05$, $p = .966$). Thus there was no testing effect due to practice.

Students did show significant gains from pre-test to post-test, with both textbook and multimedia, but more so with multimedia. Scores on the first post-test were higher for those using multimedia (mean of 14.50) than those using the textbook (12.46), a significant difference ($t = -3.337$, $df = 76$, $p<.001$). Moreover, the multimedia adds to what they learn from the textbook. Students using the textbook first, then using the multimedia improved their scores on the second post-test (12.46 to 15.40), a significant improvement ($t = -6.527$, $df = 34$, $p<.001$). We infer that the multimedia helps students learn conceptual material better than if they were exposed only to the textbook. On the other hand, there was little improvement for students using the multimedia first, then the textbook (from 14.73 to 15.13), an insignificant difference ($t = -.993$, $df = 29$, $p.<.329$). Thus it doesn't seem to matter whether the students are exposed to the multimedia before or after reading the textbook.

We conclude that many first year students can learn Java "objects-first" using BlueJ, especially with the help of interactive multimedia. We then conducted another study to determine whether we would get similar results when giving the same materials to high school students. Twenty-six pupils in Lehigh's S.T.AR. (Students That Are Ready) program (grades 10-12) participated. As with the undergraduates, the high school students improved their scores, though with smaller margins. High school students using the textbook showed modest improvement (5.31 to 7.62, $t=-3.290$, $df=12$, $p=.006$) as did those using the multimedia (from 3.23 to 9.08, $t=-6.619$, $df=12$, $p\leq.001$). The gain from pre-test to post-test was greater for students using the multimedia, averaging a gain over three points higher than the text group ($t = -3.137$, $df = 24$, $p = .004$). Multimedia improves learning more than using a textbook.

We are nevertheless concerned that the mean scores for high school students are much lower than those for undergraduates. We attribute this difference to maturity and motivation. Undergraduates understood that their work counted towards a course grade, while S.T.A.R. participants encountered this lesson without external incentive or course context, and so many expressed a lack of interest in the subject matter. We are now incorporating the multimedia for use in high schools, and also revising it to emphasize our new design-first curriculum.

4. ITS ARCHITECTURE

Hartley & Sleeman[11] proposed three components of an ITS: an expert module which contains the domain knowledge of the system, a student module which models student knowledge and behavior, and a pedagogical module which chooses appropriate teaching strategies. These three components work within the framework of a user interface that presents content and interacts

with the learner [6]. From this starting point, we developed an architecture for CIMEL ITS, shown in figure 1.

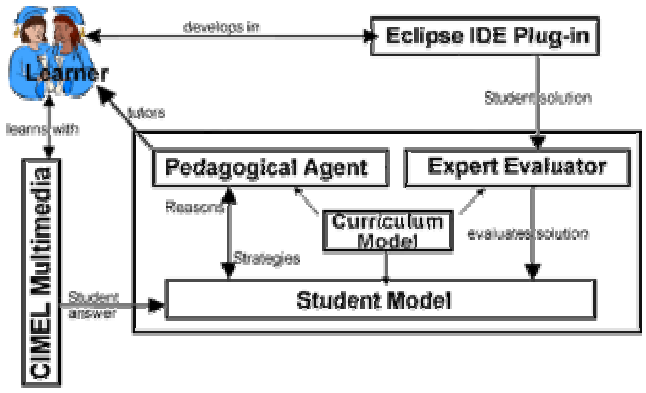

Figure 1. Architecture of CIMEL ITS

There are four components which comprise the CIMEL ITS. The heart of the CIMEL ITS is the Curriculum Model, which is a representation of the domain curriculum. The Curriculum Model is organized as a Curriculum Information Network, or CIN, which links concepts together to show relationships between them. For example, a concept may be identified as having one or more prerequisite concepts, and it may also be a component of another, higher-level concept. A difficulty measure is also assigned to each concept within the CIN. Each of the other three parts of the CIMEL ITS refers to the CIN to tie the student's learning activities and state of mind to the curriculum.

The Expert Evaluator interfaces with the Eclipse IDE through a plug-in. It observes the student's work step by step in both the object diagram interface and the code interface of Eclipse, and compares each step to its own solution(s) to the current problem. When a specific error is identified, it is linked to a concept within the CIN, and along with the recommended solution, is passed to the Student Model.

The Student Model maintains a model of the student's current knowledge state based on information from both CIMEL and the Expert Evaluator. From CIMEL, the Student Model gets input on individual student performance based on exercise and quiz data from the object-oriented contents. From the Expert Evaluator, it receives information on both errors made by the student (as described above), and problems which the student completes successfully. The Student Model then performs a diagnosis based on the history of the student's performance to determine the reasons for the student's errors and where there are gaps in his or her knowledge.

The Pedagogical Agent provides feedback to the student and tutoring when he needs help. It consists of a feedback network and tutoring strategies which may be represented by distinct agents. The feedback network is similar to CIN; it also contains feedback for each concept in the domain knowledge. Feedback is assigned a numerical level indicating if the feedback is basic or advanced. For example, the feedback consisting of concept definitions will be assigned level 1. The tutoring strategies under consideration are the traditional tutoring strategy in which an agent plays the role of a tutor and variations of cooperative learning strategies[9]. The "learning by disturbing," strategy has a traditional tutor agent and a companion agent that attempts to test student's knowledge by misleading him[1]. The "learning by teaching" strategy also has a traditional tutor agent and a companion agent learns along with the student[16].

When the student model indicates that the student needs help, the pedagogical agent selects the tutoring strategy and the feedback based on the student profile maintained in the student model. Next, the pedagogical agent interacts with the student to provide feedback / tutoring. Depending upon the strategy chosen, the student might interact with one or more agents. The traditional tutoring model consists of a single agent in the role of tutor while other strategies require one agent as a tutor and another as a companion / adversary. The feedback consists of explaining the basic concepts and relationships between various concepts, and referring the student to multimedia, relevant websites and ultimately to a human tutor.

The CIMEL ITS can interact with students either through CIMEL multimedia or the Eclipse IDE, each of which initiate different flows of control through the ITS architecture. The student can learn about object-oriented design and Java programming from CIMEL multimedia. CIMEL records the student's behaviors on the quizzes and exercises into a database. The student model uses this data to infer the student's level of knowledge. When the student has difficulties within CIMEL, the pedagogical agent[s] may intervene. Possible tutoring actions include a brief explanation of concepts that the student missed or a menu of materials that the student should review. Another possibility is walking the student through a multimedia exercise step by step, making sure the student understands how to do it and highlighting the important concepts. The pedagogical agent updates the student model with information about the instruction provided.

Another flow of control begins with the Eclipse IDE. The Expert Evaluator observes the student as he enters his design or programming solution, converts the student's solution to a representation language and compares the expert's solution with that of the student and provides the results to the student model. The expert evaluator assesses what are right answers and errors in the student's solution and the concepts tied to them. After receiving the input from the expert evaluator the student model performs a diagnosis based on the input and the history of students' records and provides the diagnosis results to the pedagogical agent. The student model considers what the current state of the student is in (struggling or not), what the student needs to know and why the student made those errors. The pedagogical agent gives proper instructions to the student based on the diagnosis results and the history of the instructions given to the student from the student model. The expert evaluator, student model and pedagogical agent each consult with curriculum model to perform their work. From the learner's perspective if the learner requests help, or appears to be struggling, the pedagogical agent may intervene based on both the student's current behavior

and his previous history as represented in the Student Model. On successful completion of the assignment, the student receives positive feedback from the pedagogical agent.

5. CONCLUSION AND FUTURE WORK

We believe that a design-first curriculum is more effective than either a structured programming or objects-first approach, because it introduces students to the big picture of software development. We have results that suggest that an objects-first approach can be effective, especially with the help of multimedia courseware. Design-first subsumes and improves upon objects-first pedagogy. Students' assignment and test results so far in both a CS1 course and a local high school have been encouraging. We will conduct formal experiments in the next semester to verify and quantify the effectiveness of our approach compared to objects-first alone.

We acknowledge that object-oriented design and programming are still challenging for students. Tools are needed to help students of all learning styles grasp the crucial basic concepts. CIMEL multimedia helps. An ITS based on the design-first approach, which works through both the multimedia and the Eclipse IDE, would fill in the gap between helping students understand the concepts and apply them in design and programming problems. We have presented an architecture and flows of control through our ITS. We plan to have a working prototype of CIMEL ITS by the summer of 2005. This will allow us to use CIMEL ITS to augment CS1 courses in the Fall 2005 semester, and to gather experimental data on its effectiveness.

6. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants No. EIA-0087977 and 0231768 and PITA (Pennsylvania Infrastructure Technology Association).

7. REFERENCES

[1] Aïmeur, E. and Frasson, C. Analyzing a new learning strategy according to different knowledge levels. *Computers in education*, vol. 27, no. 2, 1996, pp. 115-127.

[2] Allen, Eric, Cartwright, Robert, and Stoler, Brian. DrJava: A Lightweight Pedagogic Environment for Java. In *Proceedings of the SIGSCE Conference on Computer Science Education*, March, 2002.

[3] Anderson, J.R. and Skwarecki, E. (1986) The automated tutoring of Introductory Computer Programming. *Communications of the ACM*, vol. 29, September 1986, pp 842-849, ACM Press.

[4] Bailie, F., Courtney, M., Murray, K., Schiaffino, R. and Tuohy, S.. Objects First - Does It Work? *Journal of Computing Sciences in Colleges*, vol. 19, Issue 2 (December 2003), pp. 303 – 305.

[5] Barnes, D. & Kölling M. *Objects First With Java: A Practical Introduction Using BlueJ*, Englewood Cliffs, NJ: Prentice Hall, 2002.

[6] Blank, G. D., Barnes, R. F. and Kay, E. J.. *The Universal Computer: Introducing Computer Science with Multimedia* (McGraw-Hill/Primis, 2003/2004). Sample material at www.cse.lehigh.edu/~glennb/um/ and cimel.cse.lehigh.edu.

[7] Blank, G. D., Pottenger, W. M., Sahasrabudhe, S. A., Li, S., Wei, F., and Odi, H. Multimedia for computer science: from CS0 to grades 7-12, *EdMedia*, Honolulu, HI, June 2003.

[8] Bloom, B. S.. The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring, *Educational Researcher*, vol. 13, pp. 3-16, 1984.

[9] Chan T. W. and Baskin A. B. (1990). Learning companion systems. In *Intelligent Tutoring Systems: At the crossroads of Artificial Intelligence and Education* (Edited by Frasson, C. and Gauthier, G.), Chap 1. Ablex, N.J.

[10] Dag, F. and Erkan, K., (2003) Realizing of Optimal Curriculum Sequences for a Web Based General Purpose Intelligent Tutoring System, *The IJCI Proceedings* (ISSN 1304-2386, vol. 1, No 1, July 2003, International XII. *Turkish Symposium of Artificial Intelligence and Neural Networks (TAINN'2003)*.

[11] Hartley, J.R. & Sleeman, D.H. Towards more intelligent teaching systems. *International Journal of Man-Machine Studies 2,* 1973, pp. 215-236.

[12] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., The BlueJ system and its pedagogy, *Journal of Computer Science Education,* Special issue on Learning and Teaching Object Technology, vol. 13, no. 4, Dec 2003.

[13] Kumar, A. Model-Based Reasoning for Domain Modeling in a Web-Based Intelligent Tutoring System to Help Students Learn to Debug C++ Programs, *6th International ITS Conference*, Biarritz, France and San Sebastian, Spain, June 2002.

[14] McCracken M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Yifat Kolikant, Y., Laxer, C., Thomas, L., Utting, I., Wilusz, T., A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, December 01, 2001, Canterbury, UK

[15] Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B.. Constraint-Based Tutors: A Success Story. In *Proceedings of the 14th Industrial and Engineering Applications of AI and Expert Systems Conference (IEA/AIE-2001),* Budapest, Hungary, June 2001,pp. 931-940.

[16] Palthepu, S., Greer, J., and McCalla, G. (1991). Learning by Teaching. *The Proceedings of the International Conference on the Learning Sciences*, AACE, 1991.

[17] Sykes, E.R. and Franek, F.. A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. In *Advanced Technology for Learning,* vol. 1, no. 1, 2004.

[18] Ratcliffe, M. B.. Improving the Teaching of Introductory Programming by Assisting the Strugglers. *The 33rd ACM Technical Symposium on Computer Science Education*, Cincinnati, USA, March, 2002.

[19] Reiser, B.J., Anderson, J.R., Farrell, R.G.. Dynamic Student Modeling in an Intelligent Tutor for LISP Programming, *Proc. of the Eighth Int'l Joint Conf. on Artificial Intelligence*, pp. 8-14, Los Angeles, 1985.

[20] Woolf, B. and McDonald, D., Human-Computer Discourse in the Design of a PASCAL Tutor*, Proceedings of*

Conference on Human Factors in Computing Systems, Boston, MA, 1983.