Teaching Object-Oriented Software Engineering through Problem-Based Learning in the Context of Game Design

Jungwoo Ryoo
Information Sciences and Technology
The Pennsylvania State University-Altoona
Altoona, PA USA
jryoo@psu.edu

Frederico Fonseca
Information Sciences and Technology
The Pennsylvnaia State University-University Park
State College, PA USA
fredfonseca@ist.psu.edu

David S. Janzen
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA USA
djanzen@calpoly.edu

Abstract

Although Object Orientation is emphasised in software engineering education, few have attempted to alleviate the initial learning curve associated with an inexperienced audience in non-computer science disciplines. The authors propose a Problem-Based Learning curriculum centered on game development to deliver basic Object-Oriented programming concepts in an interactive and engaging manner. Class activities occur within the context of the Object-Oriented Rational Unified Process. One of the most significant contributions of this paper lies in the design of class modules containing tasks intended to educate students on Object-Oriented Software Engineering in an incremental and self-actuated way.

1: Introduction

It is often highly challenging to teach the concept of Object-Oriented Software Engineering (OOSE) to students with minimal programming experience. This statement is particularly true when the target audience of the instruction is those in disciplines other than traditional Computer Science (i.e., Information Sciences and Technology, Computer

978-0-7695-3144-1/08 $25.00 © 2008 IEEE
DOI 10.1109/CSEET.2008.26

Information Systems, Management Information Systems, etc.). These non-Computer Science majors attract many students whose ultimate goal may not be a software engineer, but learning what is involved in developing software to help them pursue other career aspects of the Information Technology (IT) industry. The students fitting this profile are less keen on studying the theories of Object-Orientation in their first day of programming class and are likely to be more enthused about the prospect of being exposed to practical, hands-on programming experiences throughout the semester, especially when the course they are taking is their first OOSE course.

There have been some significant efforts to rethink and renovate OOSE education [7, 9, 3, 4, 6] in Computer Science, but few have addressed the special needs of the non-computer science majors requiring their students to possess working knowledge in OOSE.

2: Problem Context

What challenges educators the most is often not the lack of teaching materials, but the constraints imposed upon a course. These include (1) type of audience, (2) the limited amount of time available for a prearranged set of course objectives, (3) a rigid curriculum, and (4) the choice of teaching methods.

More specifically, one of the unique set of challenges the authors face is the fact that they work at an *i*-school (or information school) where students they teach are not traditional computer science majors but a widely ranging array of technical and non-technical sub-majors whose emphasis touches upon one or more aspects of information, technology, and people, and relationships among them to promote "expertise in all forms of information, required for progress in science, business, education, and culture" [10].

The particular course (a 200 level undergraduate course entitled "Introduction to Computer Languages": IST 240) this paper focuses on is an entry level programming course that introduces Java and OOSE along with the Unified Modeling Language (UML) [8]. The main focus of the course is Java and OOSE, and the coverage of UML is expected to be rudimentary as shown in the following course objectives.

- Understand the basic principles of object-oriented programming including classes and inheritance.
- Create UML class diagrams.
- Develop, test, and execute a Graphical User Interface (GUI)-based application in Java.

The College of Information Science and Technology (IST) at the Pennsylvania State University where IST 240 is taught strongly promotes a pedagogical approach called Problem-Based Learning (PBL) [14, 5, 1, 16]. PBL is discussed in detail later in this paper (Section 3.1).

3: Approach

To cope with the challenges mentioned in Section 2, the authors combined two existing pedagogical approaches (described in the ensuing sub-sections) in a learning context

especially appealing to students. The learning context of the authors' choice is *game development* [2, 12]. Almost every student is familiar with video games and plays them. Therefore, the topic is highly tangible and engaging. Developing a game requires creativity. Creativity is somewhat stifled when a semester-long software project is given in other more conventional education settings. For example, students may be asked to develop an airline reservation system. In this scenario, requirements are provided, and students are expected to strictly adhere to the requirements. In a game development project, students can develop their own requirements for a domain in which they are experts. The creativity, however, does not deviate students from a standard set of learning goals since the games students develop are based on the mastery of a common set of OOSE skills carefully selected and incrementally presented by the authors. The common OOSE skill set covers all the major OOSE concepts as well as non-OOSE-specific, basic programming skills.

3.1: Problem-Based Learning (PBL)

PBL is a pedagogical model that emphasizes the role of a real-life problem and a collaborative discovery process in learning [14, 5, 1, 16]. Within a typical PBL setting, students are first given a challenging but realistic problem of significant size, relevant to the learning objectives of a given course. They are then encouraged to solve the problem in a group throughout the semester as independently as possible with minimum help from the instructor of the course. Apart from the traditional lecture-oriented teaching approach, PBL puts more emphasis on the instructors' role as facilitators, to prepare meaningful and interesting problems, and to create and organize course materials in a manner that students have a *just right* dose of information in each class to incrementally develop a final solution to the primary problem of the semester.

With the overarching framework of PBL, the authors partition their OOSE course into four Rational Unified Process (RUP)-like phases (i.e., Inception, Elaboration, Construction, and Transition) [13]. Students are also assigned into a group of three people to promote their maximum participation and to provide a group project experience.

3.1.1: Inception Phase PBL Activities

During the inception phase (the first four weeks of the sixteen week course), each student group is first given basic requirements. This initial set of requirements are intended to guide the students into the right direction so that the target requirements are neither too simplistic nor unreasonable for the skill sets expected to be obtained during the semester. The requirements also dictate an overall plot for the game. For example, each game shall:

- start with a main menu screen that allows end users to choose the number of players, levels of difficulty, a replay option, type of backdrops, etc.,
- have the flavor of a typical, interactive video game (players trying to either hit or avoid randomly generated objects in a scene),
- keep and display scores for each player,
- have ability to record and replay,
- have multiple scenes demanding various skill levels, and
- have sound effects.

Based on the minimum requirements above, students develop their own customized requirements. The sparse nature of the baseline requirements leaves ample room for creativity. Although requirements elicitation is the most emphasized discipline during the inception phase, students are given opportunities to refine their initial requirements throughout the semester, which is consistent with RUP. Therefore, an expected deliverable by the end of the inception phase is not a final product, but a collection of visual storyboards describing each scene to appear in the envisioned game, along with textual descriptions on the desired behaviors of individual Graphical User Interface (GUI) elements in each scene.

3.1.2: Elaboration Phase PBL Activities

The early portion of elaboration phase where analysis is emphasized is extremely abbreviated and given only a week or two since students create their own requirements as opposed to receiving a big document discussing requirements collected for an unknown domain. Therefore, the rest of the elaboration phase is mainly used for providing constructive feedback from instructors to students. The remainder of time (two to three weeks) before the construction phase is dedicated to building rapid prototypes and developing a detailed design document for the game. Although being helpful in the early stage of requirements gathering, storyboards are not as effective as having a partially working program with tangible behaviors to critique. The proposed curriculum is deliberately designed to equip students with sufficient knowledge to be able to build the rapid prototypes by the later part of the elaboration phase (sixth or seventh week of the semester). The prototypes involve:

- a fully functional initial menu window that stores and retains end user options, and
- transitions from the initial menu window to subsequent windows and proof-of-concept mechanisms to pass values between the windows.

By completing the rapid prototyping, students and instructors can at least observe the overall flow of the game program under development without worrying about the implementation details of each scene, with the exception of the main menu window. Note that the rapid prototype is not meant to be *throw-away* code, and is expected to be reused in the construction phase.

Based on their own observations and instructor feedback, students are now ready to develop a detailed design that involves making specific decisions on how each scene of the game will be implemented. Along with this information, the design details of the rapid prototype also need to be documented using a design language like Unified Modeling Language (UML). As a result, the documentation process must be preceded by a class session on a modeling language. The theme of the game and what shall be carried out concerning the contents of the game were already captured in a requirements document during the inception phase.

3.1.3: Construction Phase PBL Activities

With their instructor-approved rapid prototypes, students are in a comfortable position to smoothly switch to the construction phase. The already developed skeleton code during prototyping can now be fleshed out with the specific design details of the game (e.g., defeating space invaders, hitting a baseball, etc.). Much of this phase is led by students and run autonomously, unlike the earlier two other phases (i.e., inception and elaboration). In the first two phases, some nurturing was necessary to motivate students due to the

initial difficulty associated with learning a new programming language (i.e., Java). At the construction stage, more control over the project is intentionally relinquished to the students since they should be self-actuated by now due to the excitement resulting from the prospect of finishing their half-baked game invention with concrete visual behaviors and other effects such as sound.

3.1.4: Transition Phase PBL Activities

This phase in RUP is originally intended to ensure that migration from a development/testing environment to production is properly executed. Since the delivery of a final product to customers does not occur in a class setting, the authors simulate this process through peer reviews and demonstrations. Students review each other's work and offer their evaluations. In addition, external reviewers (instructors from the same or different disciplines, or school administrators) are invited to demonstration sessions and rate the games based on evaluation criteria provided by the instructor. These opportunities for public scrutiny seem to further increase the level of excitement and enthusiasm among students.

3.2: GUI-centered OOSE Education

The aforementioned PBL framework provides a nice vehicle for organizing a non-conventional, student-actuated, and project-oriented course. However, it does not address the problem of how to facilitate the students' self-motivated learning process with what kinds of class materials. Based on the authors' experience, this curriculum design aspect of an entry-level OOSE course is crucial in creating a highly effective PBL experience. In this section, the authors present their novel approach in lesson planning that accommodates the PBL objectives described in the previous section.

3.2.1: Laboratory Module Design

The authors' solution to the PBL-compatible, class-by-class course material delivery is the extensive use of hands-on laboratory modules. Each lab module is strategically aligned with the RUP-based PBL tasks throughout the semester as shown in Table 1. Although OOSE is explicitly dealt with only in the inception phase, it is practiced in the rest of the phases. Our view is that before creating real applications, whether it is a toy program or a functional game, it is necessary that students have at least a basic notion of OOSE. The structure the authors propose tries to address this delicate balance between doing too much theory and never creating a functional application, and creating a lot of small applications and never really understanding the theory behind OOSE.

Since GUI-based game development is the central theme of the particular PBL approach of the authors' choice, the laboratory modules are carefully designed to incrementally introduce key skills to manipulate GUI objects (such as windows, panels, buttons, etc.) in the Java programming language. Each lab module is just enough to accomplish PBL tasks for a given week. Students are also naturally exposed to OOSE concepts while obtaining their GUI skills. For example, the concept of an object and its behaviors in OOSE is not easy to understand at the abstract theory level, especially for the beginners, while the same concept is much easier to absorb when put in the context of GUI components (that is, when one can see the object like a window and observe its behavior such as minimizing, maximizing, and closing). Teaching OOSE using GUI is a well-established pedagogical approach as

demonstrated in the literature [9]. The authors' contribution is putting the proven method in an even more effective context (i.e., game development) and combining it with PBL.

4: Related Research

4.1: Alice

Alice [6] is a 3D programming environment developed to teach entry level Computer Science students basic programming concepts without exposing them to all the intricate details of a full-blown programming language. First-time programming students get easily frustrated and quickly lose their interest when repeatedly confronted with syntax or logical errors. Alice insulates students from this rather unpleasant experience by providing an error-proof Integrated Development Environment (IDE). Using Alice, students accomplish their programming tasks (usually manipulating 3D objects by calling their methods) by a series of mouse clicks and drag-and-drop maneuvers.

Unlike the Alice approach, the authors, however, tried to avoid programming environments in our course. While it is important to keep students motivated throughout the semester, the target curriculum could not afford to allot an entire semester for students get acclimated to the world of programming. There are simply too many topics to cover, and the simulated programming environment is not the same as actually dealing with a programming language. In addition, "programming environments are often either overly complex, incomplete in their language support, or do not provide good support for the teaching and learning processes, thus hindering active assignment work early in the course according to Kölling and Barnes [11]."

5: Conclusion

This paper proposed a novel approach in teaching OOSE through PBL and game development. The approach has been successfully applied repeatedly over the course of multiple years. Student responses have been very positive in general and have been quantified via generic course evaluations. The authors intend to do more focused surveys in the future to investigate exactly what aspects of the proposed curriculum are more efficient than others and why, through a thorough data analysis. Another possibility of even further improving the course design presented in this paper is incorporating robotics [4] into the curriculum. This may further engage the students and make the process of learning OOSE even more interesting.

References

[1] Howrd S. Barrows. A taxonomy of problem-based learning methods. *Medical Education*, 20:481–486, 1986.

[2] Jessica D. Bayliss and Sean Strout. Games as a flavor in CS1. In *Proceedings of the 37th ACM SIGCSE Technical Symposium on Computer Science Education*, pages 500–504, Houston, Texas, USA, March 2006. ACM.

[3] Byron Weber Becker. Teaching CS1 with karel the robot in java. *ACM SIGCSE Bulletin*, 33(1):50–54, March 2001.

[4] Joe Bergin. Karel universe drag and drop editor. *ACM SIGCSE Bulletin*, 38(3):307–307, September 2006.

[5] David Boud. *The Challenge of Problem Based Learning*. Routledge, August 1997.

[6] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, Kevin Christiansen, Rob Deline, Jim Durbin, Rich Gossweiler, Shuichi Kogi, Chris Long, Beth Mallory Steve Miale, Kristen Monkaitis, James Patten, Jeffrey Pierce, Joe Schochet, David Staak, Brian Stearns, Richard Stoakley, Chris Sturgill, John Viega, Jeff White, George Williams, and Randy Pausch. Alice: Lessons learned from building a 3d system for novices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 486–493, The Hague, The Netherlands, April 2000. ACM.

[7] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. In *Proceedings of the 34th ACM SIGCSE Technical Symposium on Computer Science Education*, Reno, Nevada, USA., February 2003. ACM.

[8] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, 3rd edition, September 2003.

[9] Jesse Heines and Martin Schedlbauer. Teaching object-oriented concepts through gui programming. In *Proceedings of the 11th Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts*, Berlin, Germany, July 2007.

[10] iSchool Deans. What are ischools? `http://www.ischools.org/oc/field.html`.

[11] Michael Kölling and David J. Barnes. Enhancing apprentice-based learning of java. In *Proceedings of the 35th ACM SIGCSE Technical Symposium on Computer Science Education*, pages 286–290, Norfolk, Virginia, March 2004. ACM.

[12] Michael Kölling and Poul Henrikson. Game programming in introductory courses with direct state manipulation. In *Proceedings of the 10th ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 59–63, Monte de Caparica, Portugal, June 2005. ACM.

[13] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 3rd edition, December 2003.

[14] Lisa Lenze. What is problem-based learning?, 2007. `http://pbl.ist.psu.edu/pbl/`.

[15] Barb Moskal, Deborah Lurie, and Stephen Cooper. Evaluationg the effectiveness of a new instructional approach. In *Proceedings of the 35th ACM SIGCSE Technical Symposium on Computer Science Education*, pages 75–79, Norfolk, Virginia, USA, March 2004. ACM.

[16] John R. Savery and Thomas M. Duffy. Problem based learning: An instructional model and its constructivist framework. Technical report, The Center for Research on Learning and Technology (CRLT), June 2001.

Week	PBL Tasks	Programming Language Concepts	GUI Topics	Relevant OOSE Concepts
1	Inception	Object-Oriented Programming Language, Virtual Machine, Byte Code, Compiler		Classes, Objects, Instantiation, Attributes, Methods, Constructor
2		Conditional Statements		Calling a method in another method
3				Referencing an object in another object
4				Inheritance
5	Elaboration	Looping	Containers and Components, JFrame, JMenu, JPanel, JButton, JLabel	
6		Modeling Languages, Unified Modeling Language (UML)	Layouts	
7		Rapid Prototyping	ActionListener	
8			JTextField, JSlider	Creating one object in the other object and passing a reference to the newly created object to another
9	Construction		remove(), removeAll(), Null Layout, TabbedPane	
10			MouseListener, MouseMotionListener, KeyListener	
11		Mark-Up Language, Extensible Mark-Up Language (XML)	Timer, Progress Bar	
12			XML Encoder, XML Decoder	
13	Transition			
14				
15				
16				

Table 1. Lab Modules and their Relationship to Programming Language/OOSE/PBL Learning Objectives