Team-Based Pedagogy for CS102 Using Game Design

Yolanda A. Rankin* Northwestern University
Tom Lechner† Northwestern University
Bruce Gooch‡ University of Victoria

Abstract

Computer science teachers often struggle with design programming assignments that both introduce students to object-oriented design and provide meaningful opportunities for students to develop object-oriented programming skills. As a result, teachers seek creative alternatives for educating the next generation of computer scientists. Using the context of game design, we identify a team-based pedagogical strategy to assist students with understanding object-oriented design principles. Team-based pedagogy includes well-defined rubric, application-focused team programming assignments, software development tools and built-in incentives for individual and team learning. We extend Microsoft's Flight Simulator X game platform as a software development tool for team programming assignments that reinforce object-oriented game design. Additionally, we develop team-based learning tools that promote social interactions as a part of the learning process as students develop proficiency in object-oriented programming. Finally, we propose a method for evaluating team-based pedagogy applicable to computer programming courses that influence positive learning outcomes for students.

CR Categories: K.3.1 [Computers and Education]: Computer and Information Science Education—Computer Science Education; K.3.2 [Computers and Education]: Computer Uses in Education—Collaborative learning; D.1.5 [Software]: Coding Tools and Techniques—Object-oriented programming

Keywords: Computer science education, game development, object-oriented programming, computer supported collaborative learning

1 Introduction

Education is a crucial component for training and producing the next generation of computer scientists. Traditional core computer science curriculum focuses on acquiring expertise in software development with proficiency in one or more programming languages applicable to a particular specialty area, e.g. artificial intelligence, systems, and computer graphics. Teachers struggle to design programming assignments that both introduce students to software engineering concepts and provide ample support for beginning programmers; these programming assignments often fall short of helping students to adopt object-oriented design principles including modeling, design code analysis, testing and debugging of standalone modules, and the application of computer science principles to solving real-world problems [Alphonce and Ventura 2002; Feldman and Zelenski 1996]. Currently, enrollment in Computer Science programs across the nation has dropped 60% and the trend forecasts a continued decrease [Snyder 2006; Vesgo 2005] . Such statistics have sounded the alarm in computer science departments across the nation. If we hope to attract and retain computer science students, then we must focus on de-mystifying the art of software design and give students a strong foundation upon which to develop their programming skills.

To combat the declining interest in computer science in recent years, Guzdial & Soloway [2002] recommend leveraging generation X's love affair with digital media as a means to generate interests in courses perceived to be designed exclusively for techies, geeks and programming gurus. In response, computer science departments across the country have adopted the digital medium of computer games as a context for software development [Alphonce and Ventura 2002; Bayliss and Strout 2000; Johnson et al. 1998; Jones 2000; Michaelson et al. 2004; Parberry et al. 2005; Parberry et al. 2006; Rankin et al. 2007]. Teachers combine students' familiarity and recreational enjoyment of computer games as a pedagogical strategy for reviving their computer science curriculums. They accomplish this goal by using the specialty area of game design as the infrastructure for beginner, intermediate, advanced computer science students to develop object-oriented design skills.

Previous research efforts suggest that a tighter collaboration is needed between the academic preparation of computer science students to pursue careers in industry and actual software development practices [Parberry et al. 2005; Parberry et al. 2006; Rankin et al. 2007]. However, more emphasis has been placed on incorporating industry tools for game design in the classroom than proper assessment of pedagogical strategies. While we agree that use of industry tools is an important component to aligning academia with game industry practices, current game-oriented curricula has failed to place proper emphasis on the importance of teamwork as both a part of the learning process and as a required industry practice for upcoming software developers. We argue that effective pedagogical strategies in the domain of computer science must include tools for software development as well as tools for collaborative learning. Thus, we extend a game development kit and leverage computer mediated communication tools to accommodate team-based learning.

Positioning teamwork as the basis for learning-by-doing, we introduce the concept of team learning in the context of game development. Using Microsoft's Flight Simulator X (FSX) as the software development platform, we define team learning pedagogical strategies in an introductory computer science course. In addition, we develop web-based tools that scaffold novice programmers and apply pedagogical strategies that foster collaborative learning as students acquire advanced programming skills. Finally, we discuss the implications of team learning on individual assessment of object-oriented design principles.

*e-mail: yrankin@northwestern.edu
†e-mail:tel768@northwestern.edu
‡e-mail:brucegooch@gmail.com

2 Pedagogical Strategies for Team-Based Learning

According to Edelson et al. [1995], the act of communicating ideas and concepts to others enables individuals to construct new knowledge that is closely linked to social interactions with other students. These same social interactions serve as the adhesive force for creating a community of learners who participate in shared practices [Lave and Wenger 1991; Scardamalia and Bereiter 1996; Wenger 1999]. Consequently, learning environments embrace interactive media as a communication tool for sharing information and giving students access to communities of learning [Bruckman 1998; Forte and Bruckman 2006]. The exchange of information happens naturally in a collaborative learning environment. Collaborative learning is transformed into team-based learning when students divide into smaller groups with each group responsible for completing a specific task that contributes to the realization of a pre-determined final product. We define team-based learning as a superset of collaborative learning in that students are intentionally divided into groups for the purpose of acquiring knowledge applicable to completing a task and/or assignment; each group member practices positive social interactions and portrays an important role (e.g. project manager, artifact producer, knowledge sharer) to meet learning objectives [Edelson et al. 1995; Kwon and Gomez 2004; Soller 2001]. As a consequence of group social interactions, team members function as the first tier of support for collaborative learning by provoking members to propose ideas, raise questions, construct knowledge and engage in reflective thinking [Salomon et al. 1991]. Borrowing heavily from Michaelson et al. [2004], we identify key elements of team-based learning as:

- Readily available software development resources
- Well-defined rubric for group programming assignments
- Incentives for individual programming assignments
- Application-focused programming team assignments

These four criteria represent the requirements for designing interactive, team-based tools in the context of game design.

2.1 Accessible Resources

Readily available resources aid students in completing assignments and produce significant learning outcomes. Resources ought to consist of easily accessible software development tools that allow students to partner with technology and other students so that the work is a joint effort [Salomon et al. 1991]. Such partnerships can result in more intelligence than is humanly possible alone, creating ideal cognitive conditions for students to enter "the zone of proximal development" [Vygotsky 1978]. The zone of proximal development refers to what the student knows and the student's potential knowledge as a result of social interactions with more capable others [Vygotsky 1978]. We believe that team-based learning maximizes opportunities to socialize with students of various levels, resulting in more creativity, greater productivity, and increased cognition such as quicker acquisition of knowledge. However, instructors must be careful that software development tools are used mindfully, that they challenge the student, and that the student is able to use them effectively without becoming distracted from the original intent of the course. In light of the goal of team-based learning, teachers play a pivotal role in facilitating the learning process, serving as expert guides that introduce the students to game development tools. As students learn to use these game development tools, they progress from peripheral participation into fluent users of technology. Therefore, students become acculturated into the computer programming community [Bruckman 1998; Edelson et al. 1995; Kwon and Gomez 2004; Lave and Wenger 1991; Wenger 1999].

2.2 Well-Defined Rubric

Rubric explicitly outlines assignments, learning objectives and explains criteria for individual and group assessment. A well-defined rubric is a necessity if students are expected to meet learning expectations and encourage team work [Kwon and Gomez 2004]. An effective pedagogical strategy requires instructors to communicate clearly what each assignment entails, how students will be evaluated on an individual and team basis and what students should know as a result of taking the course. More importantly, technology that places the teacher's expectations and learning goals in a public place serves as a checklist for team programming assignments, initiates discussions among team members and invites further clarification from students. Learning tools that communicate course rubric typically take the form of computer mediated communication, including chat rooms, online discussion forums, electronic notebooks, and course websites [Bruckman 1998; Edelson et al. 1995; Forte and Bruckman 2006].

2.3 Incentives for Individual Learning

Although team-based learning emphasizes collaboration among group members to acquire object-oriented design skills, it is not a substitute for individual students mastering these skills. Computer science teachers need to incorporate tools that provide built-in incentives for individual preparatory work to supplement individual learning and increase team productivity [Trytten 2005; Trytten 2001]. Team-based learning tools are designed to showcase individual contributions. For example, in the context of game development, an ideal team-based learning tool gives each student code templates; code templates focus the student on solving the problem and ignoring extraneous details [Alphonce and Ventura 2002; Feldman and Zelenski 1996]. Tools should accommodate students displaying code to the team and the class as a whole. In order for the team to be productive, each student is accountable for developing a portion of the code that contributes to the team's final product.

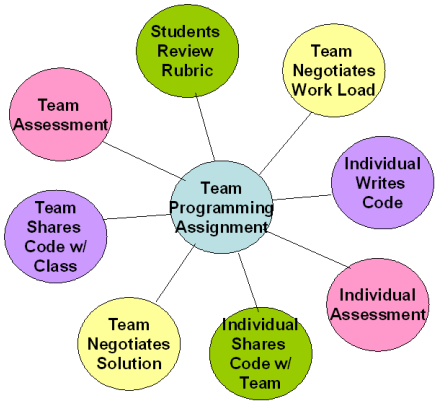

Figure 1: Learning model for Team-based Application Programming.

2.4 Application-focused Team Programming

Application-focused team programming assignments engage students in social interactions that promote team cohesiveness and group learning. Application-focused programming assignments require students to construct game artifacts (e.g. background scenery, and characters) that are shared within the community of learners. To foster social interactions which force students to form teams, the programming assignment must be a substantial project that is impossible for one person to complete given specific time constraints. As teams participate in a top-down design approach to solving the programming assignment, they engage in supportive communicative practices such as asking for assistance, sharing information that builds group knowledge, acknowledging peers ideas, mediating conflict when group members disagree, and providing positive feedback [Kwon and Gomez 2004; Soller 2001]. Applying these elements of team-based learning to the domain of introductory object-oriented programming, we construct a collaborative software development environment that leverages the appeal of computer games while simultaneously assisting students with acquiring and applying new object-oriented programming concepts. Figure 2 demonstrates the learning model for application-focused team programming assignments.

3 Implementation of Team-Based Learning Modules

3.1 Microsoft FSX: Modified Game Development Platform

Microsoft presents the latest in entertainment and aviation education in its release of Flight Simulator X (FSX). A culmination of nearly 25 years, FSX invites the player to step into the cockpit and fly to any destination of her choice. Used for pilot training and high school education, FSX offers stunning graphics and a realistic flying experience for players of various skill levels. Players learn how to successfully pilot aircraft and direct air traffic. The in-game Learning Center provides step-by-step tutorials for pilots to learn how to apply physics and math concepts to successfully navigate aircraft and complete flight missions. The game play experience familiarizes players with game controls and introduces knowledge required to successfully pilot an aircraft and direct air traffic.

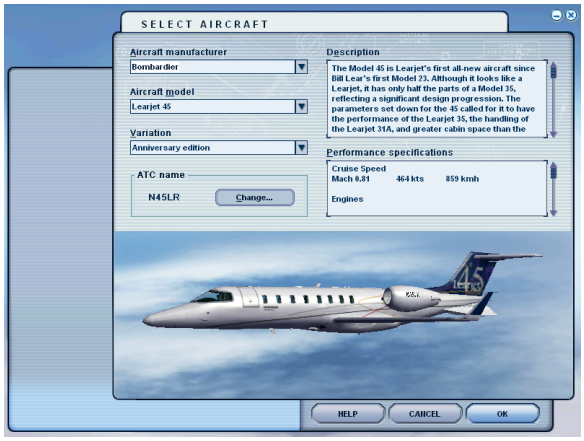

Figure 2: FSX Model of Bombardier Jet Aircraft.

We modify the FSX SimConnect API to be a manageable game platform development tool that provides adequate support for designing game add-ons without overwhelming novice programmers with complex software. We develop and integrate additional software development tools to help students design a game module featuring an innovative recumbent bike interface. See Figure 3.

Figure 3: Gforce recumbent bike interface for innovative game-play experience.

The recumbent bike corresponds to piloting a specific aircraft, giving players the benefit of exercise and entertainment. For example, when a player attempts to take off, the player must pedal fast in order for the aircraft to reach the appropriate speed for take off. The player's heart rate increases as the player pedals faster. Piloting a plane at an appropriate speed helps the player to maintain target training zone for heart rate. Hence, students are responsible for designing a game module that motivates users to workout while learning how to fly. Using the FSX game platform, team projects culminate into a flying bike game module, creating a unique game-play experience for players. Thus, we transform FSX into an artifact that fosters team learning. Ultimately, FSX serves as a test harness for each team project and the class final project—a flying bike game module.

3.2 C# Library for Flight Components

Understanding that students are writing code in the context of gaming, we ask students to research and identify the components of a flight mission. The components translate into objects that interact with one another to create the game-play experience. Students examine game design principles that enhance game-play experiences, delve deeper into the relationships that exist between game resources (i.e. aircraft, gauges, haptic feedback) and design elements (i.e. conflict, challenge, fun) that influence a positive experience for players.

Using an objects-first approach, students first identify game objects required to complete a mission and game objects that supply a virtual environment. Emphasis is placed on breaking the problem into manageable pieces by identifying what is needed to design a flight mission. Students then explore the existing relationships between these game objects, participating in the initial phase of game design. We build a C# library of data structures (arrays, linked lists, trees, etc.) to manipulate, manage, and construct game objects and resources that allow students to become familiar with the representation of handling data structures before learning how to implement them [Alphonce and Ventura 2002]. The C# library serves as

the foundation for assisting novice programmers to develop their object-orientation design skills as individuals and as a team. For example, we define a class of aircraft that includes aircraft components such as wings, rudder, and gauges. The class of aircraft components is similar to existing FSX aircraft components and serve as the basis for any aircraft's mechanical design and flight dynamics. Computer science students in an introductory object-oriented design course will extend the library, designing and implementing objects required for the programming assignment.

3.3 Object-oriented Game Design Wiki

A wiki is a website that features a deposit of information about a particular topic. What makes a wiki different from other reference websites is that anyone can edit its content. Thus, wikis empower users, enabling them to add, delete or edit content [Forte and Bruckman 2006]. One of the drawbacks is that the information may not be correct. While users may post incorrect information to the wiki, other users can edit inaccurate information, helping to build an accurate knowledge base. Empowering students to provide feedback to peers, share information, and assist with team-learning, we create an object-oriented game design wiki that accommodates individual and teams acquiring knowledge of object-orientation design principles. The wiki functions as a pedagogical tool that promotes teachers as facilitators that help students master object-oriented programming concepts and that encourage collaborative learning practices, such as sharing ideas, code, and helpful tips while responding to their peers' request for assistance. Students learn how to upload files and images to the wiki. The wiki affords students the opportunity to engage in social interactions that enable the learning process using the chatroom; the exchange of information among students helps the instructor to identify the concepts students understand and common misconceptions [Forte and Bruckman 2006]. The wiki positions teachers as expert guides in the game design process and assist students with becoming legitimate participants in the community of game developers. Additionally, the wiki publicly displays the criteria for individual and team-based project assignments. As a result, the wiki serves as the cornerstone for building a learning environment that transforms entry level computer science students into proficient programmers.

3.4 Application-Focused Programming Exercises

The primary objective of the first assignment is for students to become proficient users of FSX. Students learn how to successfully pilot an aircraft of their choice and successfully navigate air traffic. Additionally, students are initiated into the game development process by evaluating the game-play experience of FSX. Once students have had an opportunity to become familiar with piloting various aircraft using FSX, the class will select one aircraft for design and implementation as the application-focused programming assignment and ultimately design a game module to be tested by their peers and high school students in the local community. The SimConnect API for Flight Simulator X provides these novice programmers with the foundation to master object-oriented programming concepts and further hone their emerging software development skills as it applies to game development. The wiki outlines seven laboratory modules relevant to game design and object-oriented programming concepts. The entire class will collaboratively design and implement game components, using the aircraft class defined in the C# library. Each module identifies learning and programming objectives, and students gain hands-on experience writing C# code that produces game artifacts (e.g. aircraft, ground vehicles, weather conditions, airports, etc.). Each team is responsible for implementation of a specific plane component. For example, one student may choose to design the wings while another student develops code that operates the attitude indicator instrument and displays the necessary information on the panel. To complete this programming assignment, each member of the team must work together to produce a team product—a flying bike interface that players can navigate using the instrumentation panel.

Figure 4: Template of cockpit for Curtiss Jenny aircraft stubbed for gauges.

3.5 C# Interface for Team Programming Assignments

The Microsoft FSX game engine comes equipped with a documented API designed with the intent for users to add their own modules to the game environment. Most portions of the library can be extended using C#, though there are portions that can be only extended through XML and C++, such as the gauge extensions. Additionally, the API is highly advanced and in its original state it would be overwhelming for a set of novice computer programmers. For this reason we have created wrapper classes that encapsulate the key components of the SimConnect library, and provide a simplified interface for the team projects. These classes are designed to abstract the technical internals of the game engine and allow students to focus more on creativity, design, and programming practices.

We also provide students with a set of generic data collection interfaces with the intention that students be immediately confronted with the distinction between interfaces and their respective implementations. Students will have the responsibility to implement a subset of data structure types, such as vectors vs. linked lists, and will then be able to demonstrate the impact different algorithms have on a program, even when the interface remains the same at a higher level. These data structures will be incorporated into the rest of the team projects by being directly accessed by the C# wrapper library. (See Figure 4.) These interfaces are designed to encourage team-learning by first demonstrating the checks and balances considered in designing algorithms, and then by determining the circumstances where particular design decisions are most effective. Given that the number of permutations of types of data structures is quite large, each student can be given his own particular assignment without needing to work on the same assignment as one of his peers. This can reduce the temptation to copy or fake work, while still providing enough similarities to foster assistance and communication with the rest of the class.

Figure 5: FSX Prototype of Flying bike interface.

4 Discussion & Future Work

Using the context of game development, we have identified team-based pedagogical strategy for assisting students in an introductory computer science course with developing proficient object-oriented programming skills. The goal is for students to acquire object-oriented design techniques and experience developing a game module. However, careful evaluation is needed if we hope to see positive learning outcomes as a result of implementing team-based pedagogy.

Future research examines the implications of team-based learning and its effect on individual attaining learning objectives. We propose a methodology for evaluating the effectiveness of team-based learning tools on group and individual acquisition of object-oriented programming skills. To measure its effect, we need to compare the quality of individual code to team-produced code, evaluating code for evidence of object-oriented design techniques such as encapsulation, polymorphism and inheritance. How does team programming impede individual learning? How is it beneficial in core computer science curriculum? Before we can presume that team-based learning is a viable option for teaching programming courses, we will need to compare the learning outcomes of students enrolled in a traditional, introductory computer science course to students who complete a team-based programming class. Students will complete course surveys that measure the impact of team-based learning on students choosing computer sciences as a major and students' perceived learning outcome. Furthermore, we evaluate team-based pedagogical tools use for game design. Which tools support team-based game development and why? Which ones do not and why? We will collect metrics for frequency of use, time spent on task, and usefulness for individual and team programming. These results will inform team-based pedagogy in the classroom setting and lead to positive learning outcomes for computer science students.

5 Conclusion

We identify a team-based pedagogical strategy for assisting students with acquisition of object-oriented design principles in an introductory computer science course. Team-based pedagogical strategy organizes students into groups where each group has a specific programming assignment to complete. In order for the group to successfully complete the assigned task, each group member engages in social interactions as a means for acquiring both individual and group programming skills. Using the context of game design, we modify Microsoft's Flight Simulator X platform to introduce students to object-oriented design concepts and provide a development environment for team programming assignments. In addition, we design team-based learning tools that promote team members' social interactions as a part of the learning process while both individuals and teams acquire proficiency in object-oriented game design. We propose a method for evaluating team-based pedagogy and its effect on individual learning in computer programming courses. Finally, we express our appreciation to Microsoft Research for their generous support.

References

AHARONI, D. 2000. Cogito, ergo sum! cognitive processes of students dealing with data structures. In *Proceedings of SIGCSE 2000 Technical Symposium on Computer Science Education, March 2000. Austin, TX*, 70 – 74.

ALPHONCE, C., AND VENTURA, P. 2002. Object orientation in cs1-cs2 by design. In *In Proceedings of 7th Annual Conference on Innovation and Technology in Computer Science Education*, ACM Press, 70–74.

BAYLISS, J., AND STROUT, S. 2000. Games as a flavor of cs1. 500–504.

BRUCKMAN, A. 1998. Community support for constructionist learning. In *The Journal of Computer Supported Collaborative Work*, vol. 7, 47–86.

EDELSON, D., PEA, R., AND GOMEZ, L. 1995. Constructivism in the collaboratory. Englewood Cliffs, NJ. Educational Technology Publications.

FELDMAN, T. J., AND ZELENSKI, J. D. 1996. The quest for excellence in designing cs1/cs2 assignments. In *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, Philadelphia, PA*, ACM Press, 319232.

FORTE, A., AND BRUCKMAN, A. 2006. From wikipedia to the classroom: Exploring online publication and learning. In *Proceedings of the 7th International Conference on Learning Sciences, Bloomington, IN*, 182188.

GUZDIAL, M., AND SOLOWAY, E. 2002. Teaching the nintendo generation to program: Preparing a new strategy for teaching introductory programming. In *Communications of the ACM*, vol. 45.

JOHNSON, D., JOHNSON, R., AND SMITH, K. 1998. Cooperative learning returns to college. In *Change*, vol. 30.

JONES, R. 2000. Design and implementation of computer games: a capstone course for undergraduate computer science education. In *In Proceedings of te 31st SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 260 264.

KWON, S. M., AND GOMEZ, L. 2004. Strengthening learning communities by promoting social skill development. In *Proceedings of the 6th International Conference on Learning Sciences, Santa Monica, CA*, 286 – 293.

LAVE, J., AND WENGER, E. 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.

MICHAELSON, L., KNIGHT, A., AND FINK, L. 2004. *Team-Based Learning: A Transformative Use of Small Groups in College Teaching*. Stylus Publishing, Sterling, VA.

PARBERRY, I., RODEN, T., AND KAZEMZADEH, M. 2005. Experience with an industry-driven capstone course on game programming. In *Proceedings of the 2005 ACM Technical Symposium on Computer Science Education, St. Louis, MO*, 91 – 95.

PARBERRY, I., KAZEMZADEH, M., AND RODEN, T. 2006. The art and science of game programming. In *Proceedings of the 2006 ACM Technical Symposium on Computer Science Education, Houston, TX*, 510 – 514.

RANKIN, Y., GOOCH, B., AND GOOCH, A. 2007. Interweaving game design into core cs curriculum. In *Proceedings of the 2007 Microsoft Academic Days Game Development Conference, Nassau, Bahamas, February 21 25, 2007.*

SALOMON, G., PERKINS, D., AND GLOBERSON, T. 1991. Partners in cognition: Extending human intelligence with intelligent technologies. 2 – 9.

SCARDAMALIA, M., AND BEREITER, C. 1996. Student communities for the advancement of knowledge. 36–37.

SNYDER, N. 2006. Universities see a sharp drop in computer science majors.

SOLLER, A. 2001. Supporting social interaction in an intelligent collaborative learning system. 40 – 62.

TRYTTEN, D. A. 2001. Progressing from small group work to cooperative learning: A case study from computer science. In *Journal of Engineering Education*, vol. 90, 85 – 92.

TRYTTEN, D. 2005. A design for team peer code review. In *Proceedings of SIGCSE '05, February 23 - 27, 2005, St. Louis,MO*, 455 – 459.

TURNER, J., AND ZACHARY, J. 1999. Using course-long programming projects in cs2. In *Proceedings of the SIGCSE 99 Technical Symposium on Computer Science Education, New Orleans, LA*, 43 – 47.

VESGO, J. 2005. Interest in cs as a major drops among incoming freshman.

VYGOTSKY, L. 1978. *Mind and society: The development of higher mental processes*. Cambridge: Harvard Universitiy Press.

WENGER, E. 1999. *Communities of Practice: Learning, meaning and identity*. Cambridge: Cambridge University Press.