Learning Computer Science in the "Comfort Zone of Proximal Development"

Nicole Anderson Tim Gegg-Harrison
Department of Computer Science
Winona State University
Winona, Minnesota 55987
{nanderson,tgeggharrison}@winona.edu

ABSTRACT

As computer science faculty, we are always looking for better ways to recruit and retain new majors. One way to achieve this goal is to incorporate material into the introductory courses that lies within the intersection of the student's "zone of proximal development" (which contains concepts that the student is capable of understanding) and "comfort zone" (which contains concepts that motivate the student and are presented in a way in which the student is familiar and comfortable). We refer to this region as the "comfort zone of proximal development." In this paper, we present a "comfort zone of proximal development" that we have created for computer science students which consists of a collaborative learning environment where programming concepts are introduced with gaming applications.

Categories and Subject Descriptors

[**Computing education**]: Computing education programs—*Computer science education*

General Terms

Algorithms, Design

Keywords

Active Learning, Scaffolding, Pair Programming/Teaching, Mobile Game Development

1. INTRODUCTION

Students enter the world of computer science expecting to harness new and exciting technology. When the introductory courses fail to realize their expectations, many students leave the major. Teachers want to make introductory courses enjoyable for their students, and are looking to recruit and retain new majors. However, they understand that strong fundamental knowledge must be conveyed in these courses in order for students to be successful in the field. The core concepts cannot be sacrificed. We have found a solution that provides the best of both worlds. In our approach, the introductory programming sequence is not disturbed. Instead, we developed companion courses which reinforce fundamental concepts using mobile game development and a collaborative approach, giving both students and faculty what they want.

We believe that optimal learning takes place for students when the material is within their cognitive capabilities, it is presented in a way that is familiar to them, and uses an application that is relevant or interesting to them. This is a region that lies within the intersection of the *zone of proximal development* [28] and their *comfort zone* (which contains concepts and topics that the students are interested in and comfortable with, and that are presented in a way in which they are familiar and comfortable) as depicted in Figure 1. To be effective, we need to both stay within the students' developmental zone of proximal development as well as within their comfort zone. In other words, we need to be talking about something of interest in an environment that motivates them and it needs to be concepts that the students are capable of understanding. Our notion of a *comfort zone of proximal development* is related to situated cognition (or authentic learning) [6]. Situated cognition relies on the idea that conceptual knowledge can be more easily internalized when context is provided, especially when that context is integrated with everyday life activities. According to Brown, Collins, and Duguid, "Activity, concept, and culture are interdependent. No one can be totally understood without the other two. Learning must involve all three" [6]. By relating problems to the students' lives, authentic learning occurs.

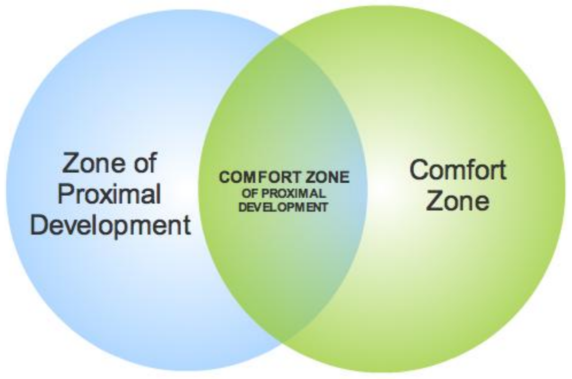

Zone of Proximal Development
COMFORT ZONE OF PROXIMAL DEVELOPMENT
Comfort Zone

Figure 1: Comfort Zone of Proximal Development

Children's playgrounds are significantly different today than they were 50 years ago. They are safer, void of "dangerous" equipment, made from various composites that provide an environment for children to play without the risk of getting hurt. That's a good thing, right? The simple answer is *yes*. Upon closer analysis, however, it is possible that the benefits in safety come at a price. Sandseter and Kennair argue that playgrounds have become too safe [27]. They see playgrounds as environments where children "test possibilities and boundaries for action," arguing that "the rehearsal of handling real-life risky situations through risky play is thus an important issue." In essence, the playground and the other children are part of the child's comfort zone. A playground with proper equipment (i.e., equipment that is safe enough but also capable of providing physical and mental challenges for children of all ages) becomes the child's zone of proximal development. Together, with an attentive parent who places restrictions when necessary, these aspects of the playground become a comfort zone of proximal development for the child. Our pair^2 learning environment that introduces computer programming through gaming applications provides an appropriate playground for individuals wanting to learn computer science. They have an opportunity to collaborate with their peers, construct useful artifacts like computer games, and challenge themselves with the security of a partner and an attentive teacher to help keep them from becoming overwhelmed. As a side-effect of "playing" in this comfortable environment, they learn computer science.

2. COMFORT ZONE OF PROXIMAL DEVELOPMENT FOR CS

Over the past several years, we have been developing a comfort zone of proximal development for our computer science students. We began by updating the discrete mathematics course to make the material more relevant to computer science students [12], and more recently changed the introductory programming sequence. The comfort zone of proximal development contains both concept-dependent and concept-independent components. The zone of proximal development is concept-dependent, requiring the selection of a concept (or topic) that is within the student's intellectual capability and one that the student is interested in so as to help motivate her to learn the new concept (or topic area). The comfort zone, on the other hand, is concept-independent, requiring the creation of a learning environment that is more conducive to student learning.

2.1 Zone of Proximal Development

Vygotsky introduced the *zone of proximal development* (ZPD) as "the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers" [28]. In other words, the zone of proximal development is what a child or individual can achieve with some guidance that they would be unable to achieve independently. The goal is to stretch the learner beyond her innate capabilities, challenging her while providing support to meet her challenges. The ZPD is unique to each individual, and varies according to her capabilities and the environment and context in which the development is occurring.

Apprenticeship and scaffolding are some common approaches to reaching the ZPD [10]. Scaffolding involves the construction of appropriate supportive conditions for a learner [11, 13]. Consider the scaffolding that is constructed for maintenance or repair of a building. It allows a worker to do jobs that they would not be capable of doing without climbing on the scaffolding. For example, scaffolding allows a worker to wash higher windows, provides a work surface for holding necessary construction tools, and often provides a level of safety and stability that gives comfort to the worker and thus the confidence to perform the job at hand. The term scaffolding may be used more abstractly in our context, but with familiar goals. Scaffolding for a learner may consist of providing the appropriate tools, support, and confidence building to allow the learner to succeed. When these types of models are employed, after spending time working with an expert to grasp a new concept an individual is able to internalize what she has learned, and eventually the scaffolding/assistance can be removed. This is reflected by Vygotsky, as he believed that "what the child is able to do in collaboration today he will be able to do independently tomorrow" [29].

In the ZPD the expert engages in a "dialogue with the novice (learner) to focus on emerging skills and abilities" [24]. Another method in which knowledge can be passed from an expert to a novice is when the novice utilizes imitation to mirror what the expert has already mastered. However, this is only effective when the action being imitated is at the appropriate developmental level. "It is well established that the child can imitate only what lies within the zone of his intellectual potential... [T]he child can enter into imitation through intellectual actions more or less far beyond what he is capable of in independent mental and purposeful actions or intellectual operations" [30]. One of the authors is the parent of a young child, and witnesses this phenomenon frequently. As an example, describing how to blow up a balloon to a child is a futile task. But while blowing up a balloon for the child, the author noted how the child began to purse his lips, puff out his cheeks, and blow out air through his mouth. Without an intended lesson, the child was imitating the actions of the "balloon blowing expert." Later, the author witnessed the child blowing up the balloon independently. The child could not inflate the balloon as large as the adult, nor could he tie the balloon to keep it inflated, but incremental development was indeed occurring to move him toward mastering these skills.

What we see is that these types of collaborations occur intentionally and unintentionally, they occur with teacher-student pairs, when mentoring and apprenticeship roles have been established, and also seemingly randomly through imitation or observation when the necessary context and environment are present. It has been noted that "the term 'collaboration' should not be understood as a joint, coordinated effort to move forward, where the more expert partner is always providing support at the moments where maturing functions are inadequate. Rather it appears that this term is being used to refer to any situation in which a child is being offered some interaction with another person in relation to a problem to be solved" [7].

As teachers, we wish to utilize these ideas to make our interaction with students as effective as possible. According to Vygotsky, teaching "is effective only when it awakens and rouses to life those functions which are in a stage of maturing, which lie in the zone of proximal development" [30]. The expert is able to empower the learner through dialogue and guided participation [26]. While some prior research encourages the teacher to attempt to find each student's ZPD, Meira and Lerman argue that the "ZPD emerges, or not, in the moment, as part of the microculture of the classroom" [17]. We believe this emergence is really the discovery of the student's *comfort zone of proximal development* (CZPD), where each individual's ZPD intersects with their comfort zone.

2.2 Comfort Zone

One of the most important factors that impacts college students' persistence is the degree of social connectedness that they experience [14, 19]. Bean argues that "few would deny that the social lives of students in college and their exchanges with others inside and outside the institution are important in retention decisions" [4]. The teacher's role in establishing the sense of connectedness is vital. Roberts and Styron argue that "effective faculty-student interaction will help establish an environment where students feel that faculty members truly care about them as individuals, which will facilitate the attainment of academic success" [25].

According to Oblinger, today's *millennial* students gravitate towards groupwork, preferring "teamwork, experiential learning, structure, and the use of technology" [21]. Ironically, however, social networking technologies appear to have a negative impact on students' social connectedness at universities. Because of the ease with which to stay "connected" with their friends from high school via FaceBook and text messaging, even those attending colleges that are geographically disconnected, today's college students do not feel the need to establish new friendships. We are finding that over half of our students will choose to have the teachers define groups rather than self-selecting partners for group work claiming that they "really don't know anyone in the class." The computer science discipline tends to attract individuals who have more introverted personality types. As such, the need to establish connectedness is both more difficult and more important for the retention of computer science students.

When properly administered, pair teaching increases both the amount and quality of teacher-student interaction. Pair programming [32] appears to be a mechanism that forces computer science students to establish a social connectedness with fellow students, something they would likely not do on their own. In addition to their social uneasiness, many of today's students have *helicopter parents* [9] with whom they have become very dependent. When pair teaching is employed, students have more opportunity to establish relationships and build stronger bonds with their teachers, which provides them with a "safety net" away from home and enables them to establish some independence and thereby feel a higher degree of connectedness to the university community. In an effort to harness the positive benefits to student learning from both pair programming and pair teaching, we recently employed *pair^2 learning*, which combines pair programming with pair teaching, in the first course of our introductory Computer Science programming sequence [1]. We believe that the combination of pair programming and pair teaching produces a learning environment that is better than when one or both of these techniques is not used, creating an appropriate comfort zone of learning for beginning CS students.

Having two teachers enables the distribution of course preparation work, placing a second set of eyes on the course material to help make it more effective. Responsibility for preparing the course materials can be divided up between topics. For each topic, one of the teachers takes on the role of the *driver*, the teacher responsible for both preparing the lecture notes and corresponding evaluations (in-class laboratory assignments, out-of-class programming assignments, and examinations), and the other teacher assumes the role of *navigator*, the teacher responsible for reviewing the course materials for defects (typos, ambiguities, and potential points of confusion). The *driver* then delivers the lecture with the *navigator* advancing the slides and demonstrating the in-class programming examples. There is constant feedback and communication between the *driver* and *navigator* during the preparation phase, delivery phase, and reflection phase. Williams argues that one of the benefits of pair programming results from the fact that "because the navigator is not as deeply involved with the design, algorithm, code or test, he or she can have a more objective point of view and can better think strategically about the direction of the work" [32]. The same applies to pair teaching. When pair teaching is employed in developing a course, the teacher's time is shifted somewhat from preparing course materials to time spent in the classroom. Of course, even though the primary responsibility for material development for a given section of the course is given to one teacher, additional time is spent by the other teacher to review the materials produced. We believe the end result is higher quality materials, which will be beneficial to the students.

2.3 CZPD for CS Students

Although it is hard to imagine anything more awe-inspiring than the closed form of the Fibonacci sequence, most of today's potential computer science students are not that impressed. As a general rule, computer science faculty members expect that every student who arrives at her university with an intention to major in computer science is excited by computation and formal reasoning. It is a reasonable assumption given that many computer science faculty members pursued advanced degrees in computer science precisely because they were awed with the beauty and elegance of computation and formal reasoning. The new generation of computer scientists, on the other hand, does not share that intrinsic love of formalism. There are other things that excite them. Towards the top of this list is their iPhone and the various gaming apps that it contains. Studies have found that students prefer gaming assignments to non-gaming assignments. Cliburn and Miller found that 78.9% of their students preferred projects that were game related, and 84% of their students claimed that games provided extra motivation to both complete their projects and improve the quality of their projects [8].

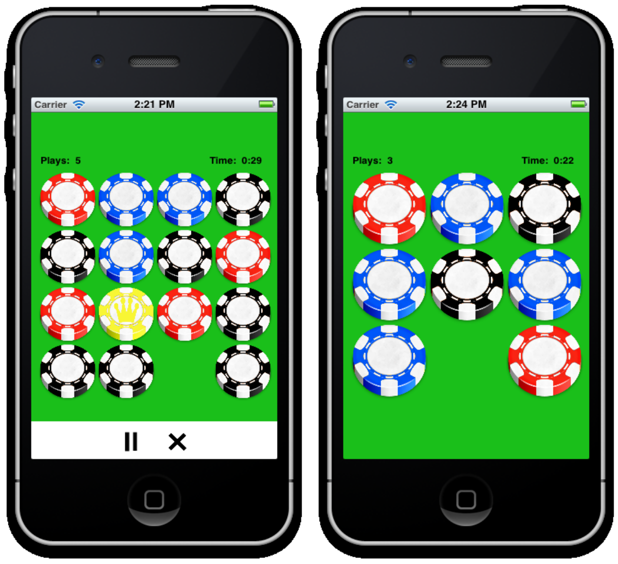
Mobile game development provides a new and exciting avenue to recruit and retain the new generation of computer scientists. There have been various approaches to incorporate game and/or mobile device programming into the computer science curriculum as a way of making the curriculum more inviting to today's college student. Several programs have restructured their introductory programming sequence courses to incorporate game development [3, 5, 16, 22, 33]. Other programs introduce students to mobile game development after they have already gained some programming experience [15], some provide game development in upper-level courses [2, 31], while still others have threaded gaming throughout their curriculum [18]. All of these approaches required significant revisions to their curriculum. Although we share their enthusiasm of using mobile game development to better engage today's students, we believe that we can achieve the same goal without completely restructuring the curriculum. We have developed an approach that captures the enthusiasm that students have with mobile devices and gaming without sacrificing the integrity of the computer science curriculum.

Rather than changing the introductory programming sequence courses to teach mobile game programming, we introduced an optional one credit companion course that augments the fundamental concepts of object-oriented programming and problem solving that are taught in the first programming course. The introductory programming course continues to introduce computer science students to object-oriented programming in Java using an objects-first approach [20], while the companion course introduces students to iPhone app development using Objective C. By using Objective C to create apps on an iPhone, the companion courses serves two primary purposes: (1) it provides students with a fun environment that increases their motivation to continue studying computer science, and (2) it strengthens their understanding of the underlying object-oriented programming concepts by reinforcing the concepts using a different programming environment.

One of the potential concerns with our approach is that we are forcing the students to switch programming languages from Java to Objective C. Although it does pose some difficulties for some of the students, we believe that the use of multiple programming languages is actually good from a pedagogical point of view. It forces students to transition from a "what do I type in" mode of thinking to the "what do I need to do" mode. Because the syntax of Objective C and Java are sufficiently different from one another, the students shift their focus from what they need to type to what they are trying to accomplish. Thus, they become more concerned with the semantics than the syntax of the programming language. This belief is supported by the results of a survey given at the end of class.

We created a set of seven iPhone labs for the companion course. The companion course meets every other week, so the students were exposed to two weeks of material in the introductory Java programming course between each lab assignment in the companion course. In the first lab, students install XCode, Apple's IDE for developing iPhone apps. The students are given a sliding puzzle iPhone app that they are able to compile and run on the iPhone simulator provided by XCode. We chose this particular app because sliding puzzles are familiar to students, it is a fairly straightforward app to implement, it has different levels (starting with a 3×3 board and going up to a 12×12 board), and it has a slight twist on standard sliding puzzle apps in that the solution to this puzzle is to find a configuration in which no two tiles of the same color are adjacent to one another.

The second lab takes the app from the first lab and has students add a `Counter` class that supports the timer and move counts that are displayed at the top of the screen. This is an example of scaffolding the app so that the students are able to produce an app without having to code the entire app from scratch. The `Counter` class is the first class that is presented in Niño and Hosch's text, so the students have already studied and produced this class in Java. The key concepts for this lab include classes, variables, and methods. None of the methods and the constructor for this class have parameters. We use this class to introduce the differences between Java and Objective C. We also talk about specification and implementation (.h vs. .m files) and the difference in compilation between Objective C and Java as well as import issues. Note that in the first lab, the students merely load XCode on their machines and run the ColorClash app. In this second lab, they *create* the ColorClash app by adding the `Counter` class. Rather than merely constructing a `Counter` class that they can create, increment, and test with various values, they are able to construct a `Counter` class that has an actual purpose. They can test the class by playing a game and checking the timer and move counts to ensure that their implementation is correct. Rather than relying on their imaginations (or suggestions provided by the instructor) that their `Counter` class is useful, they see first hand the role that it plays with the implementation of the sliding puzzle as shown in Figure 2(a).



(a) ColorClash app (b) SlidingTile app

Figure 2: iPhone Apps

In the third lab, the students create the *HelloWorld* app. Traditionally, the *HelloWorld* app is the first app that students write. Although it is nice tradition to write *HelloWorld* as the first program when learning a new programming language, it is not particularly useful for a student learning to program for the first time. It is, however, useful for a student who has already constructed a component of larger, more sophisticated app. The key concepts for this lab include an introduction to XCode projects (including the creation of an iPhone app from scratch), the declaration and allocation of objects (specifically a `UILabel`), and the invocation of methods with parameters. Students are also introduced to the Cocoa Touch API. It is important to note that we do not use *Interface Builder* in our app development, so students create and place labels (and all other *views*) programmatically. Thus, they learn about the geometry of the views along with the use of Cocoa Touch objects.

The fourth lab is another scaffolded lab. In the introductory Java programming course, students are introduced to class interaction with a maze game example. The key concept introduced with this lab is the construction of methods and constructors with parameters. Students have the concept of parameter passing reinforced by seeing the difference between the way in which parameters are expressed in Objective C vs. Java. In Objective C, parameters are labeled as compared to having unlabeled parameters defined by position in Java. Students see, however, that parameters are still defined by position in Objective C, they are just labeled as well for better readability. Parameters provide an ideal example for discussing syntactic differences and the pros and cons of these decisions. Students are reminded that the concept and underlying use of parameters is the same in both Objective C and Java. It is merely two different syntactic approaches to the same problem. Such discussion helps differentiate the key conceptual similarities from the superficial syntactic differences in the two programming languages.

The remaining labs have the students creating their own sliding puzzle. In the end, they create a 3×3 standard sliding puzzle as shown in Figure 2(b). In the first of the *SlidingPuzzle* app labs, the students start by creating an app from scratch that contains one tile which is stored as a `UIImageView`. They complete the lab by adding several more tiles at various places on the screen. The second of the *SlidingPuzzle* app labs introduces the students to event handling through the `touchesBegan`, `touchesMoved`, and `touchesEnded` methods. Students practice with conditionals to identify when the user is touching one of the tiles. Students begin by tracking the finger movement precisely by updating the x and y coordinates of the tile to match the x and y coordinates of the touch. They then modify their code to restrict moving the tiles to strict vertical and horizontal movement. This lab activity is useful at this stage in the student programmer's development because it eliminates any lingering belief in the *magic* of computing, since they see firsthand that what is displayed on the screen is dictated by their code, not the actual finger movement. In the final *SlidingPuzzle* app lab, students complete the sliding puzzle by creating an array of tiles.

Scaffolding in the initial labs is at the code implementation level, where students are given a partial program solution and are asked to complete the app by writing the missing code. This implementation-level scaffolding is important for beginning programmers because they lack sufficient programming experience and knowledge to construct a complete solution on their own. Scaffolding in the final labs is at the design level, where students are guided through the design steps for the program solution and are asked to complete all of the coding. This design-level scaffolding is important for beginning programmers as they transition into more independent programmers.

3. RESULTS AND CONCLUSION

We recently offered two sections of the first course in the Computer Science introductory sequence along with the companion courses, where both sections were pair taught. We invited students to complete a survey that asked them to give some feedback on how pair teaching was influencing them in terms of learning, attitudes, and likelihood to continue on in the Computer Science course sequence. The survey included some questions they answered using a modified Likert scale with four options. The neutral option was removed so that students were forced to choose either a positive or negative response. It also included some open-ended questions on the benefits and problems students experienced from having a course that was pair taught. Finally, the survey asked students whether or not they planned to continue with the next course in the CS introductory sequence, and if their experiences in the current course influenced this decision.

The results of the survey indicated that students liked the the use of scaffolding with gaming applications in a pair^2 learning environment, with 92% indicating that they "looked forward to coming to class each day" and 29% indicating that they "greatly looked forward to coming to class each day." They also felt like they benefitted from the pair teaching with 82% of students indicating that they received "more individual attention" due to the pair model and 29% indicating that they received "a lot more individual attention" than they would have received with a single teacher. The students' positive experience in the classroom lead to higher retention, with 49% of them continuing to the next course in the sequence. This represented a significant improvement in our retention rates. The average retention rate (which is calculated based on the number of students who continue into the second programming course in the following semester) was 39% for the previous four semesters when the pair^2 learning environment was not used. We have enjoyed a high success rate in the second course in the CS introductory sequence, with 82% of our students receiving a passing grade over the past four semesters. We continued that success, with 91% of the students that participated in the pair^2 learning environment receiving a passing grade in the follow-on course (that was not taught by either of the instructors who taught the first course). These results support the claim that our approach improved retention without sacrificing student learning outcomes.

Reynolds recently noted that "bringing a spirit of play to work - and the feeling of exploration and discovery that it instills in the moment - improves learning and stimulates creative thinking" [23]. We believe that play for a child or an adult, as well as specifically for a computer science student includes two key factors. The first is exploring something that is relevant and interesting to the individual, and the second is doing this in a social environment. We have found a formula that incorporates both of these factors into our introductory CS course by using game development as our application area and utilizing pair^2 learning to create a truly collaborative learning environment. This formula has produced successful results, both in terms of student feedback and in terms of student retention in the CS program.

4. REFERENCES

[1] N. Anderson and T. Gegg-Harrison. Pair^2 learning = pair programming $\times$ pair teaching. In *WCCCE 2012*, pages 2–6, Vancouver, Canada, May 2012.

[2] T. Barnes, E. Powell, A. Chaffin, A. Godwin, and H. Richter. Game2Learn: Building CS1 learning games for retention. In *ITiCSE 2007*, pages 121–125, Dundee, Scotland, June 2007.

[3] J. D. Bayliss and S. Strout. Games as a “Flavor” of CS1. In *SIGCSE 2006*, pages 500–504, Houston, Texas, March 2006.

[4] J. P. Bean. Nine themes of college student retention. In A. Seidman, editor, *College Student Retention*. Praeger, 2005.

[5] H. Boudreaux, J. Etheridge, and T. Roden. Adding handheld game programming to a computer science curriculum. In *GDCSE 2008*, pages 16–20, Miami, Florida, March 2008.

[6] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational Researcher*, 18(1):32–41, 1989.

[7] S. Chaiklin. The zone of proximal development in Vygotsky’s analysis of learning and instruction. In *Vygotsky’s educational theory in cultural context*. Cambridge University Press, 2003.

[8] D. C. Cliburn and S. M. Miller. Games, stories, or something more traditional: The types of assignments college students prefer. In *SIGCSE 2008*, pages 138–142, Portland, Oregon, March 2008.

[9] F. W. Cline and J. Fay. *Parenting with Love and Logic: Teaching Children Responsibility*. Pinon, 1990.

[10] A. Collins, J. S. Brown, and S. E. Newman. Cognitive apprenticeship: Teaching the craft of reading, reading, and mathematics. In L. B. Resnick, editor, *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, pages 453–494. Lawrence Erlbaum, 1989.

[11] R. Donato. Collective scaffolding in second language learning. In *Vygotskian approaches to second language research*. Ablex, 1994.

[12] T. S. Gegg-Harrison. Constructing contracts: Making discrete mathematics relevant to beginning programmers. *ACM Journal of Educational Resources in Computing*, 5(2):3:1–3:28, 2005.

[13] P. M. Greenfield, B. Rogoff, and J. Lave. A theory of the teacher in the learning activities of everyday life. In *Everyday cofnition: Its development in social context*, pages 117–138. Harvard University Press, Cambridge, Massachusetts, 1984.

[14] G. D. Kuh, J. Schuh, E. Whitt, R. Andreas, J. Lyons, and C. Strange. *Involving Colleges: Successful Approaches to Fostering Student Learning and Personal Development Outside the Classroom*. Jossey-Bass, 2000.

[15] S. Kurkovsky. Engaging students through mobile game development. In *SIGCSE 2009*, pages 44–48, Chattanooga, Tennessee, March 2009.

[16] S. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. In *SIGCSE 2007*, pages 115–118, Covington, Kentucky, March 2007.

[17] L. Meira and S. Lerman. The zone of proximal development as a symbolic space. *Social Science Research Papers*, 1991.

[18] B. B. Morrison and J. A. Preston. Engagement: Gaming throughout the curriculum. In *SIGCSE 2009*, pages 342–346, Chattanooga, Tennessee, March 2009.

[19] D. Moxley, A. Najor-Durack, and C. Dumbrigue. *Keeping Students in Higher Education: Successful Practices and Strategies for Retention*. Kogan Page Limited, 2001.

[20] J. Nino and F. A. Hosch. *Introduction to Programming and Object Oriented Design Using Java*. John Wiley and Sons, 3rd edition, 2008.

[21] D. Oblinger. Boomers, gen-xers, and milliennials: Understanding the new students. *Educause Review*, 38(4):37–47, 2003.

[22] R. Rajarvivarma. A games-based approach for teaching the introductory programming course. *SIGCSE Bulletin*, 37(4):98–102, 2005.

[23] G. Reynolds. The secret to great work is great play. *http://www.presentationzen.com/presentationzen/2010/03/we-were-born-to-play-play-is-how-we-learn-and-develop-our-minds-and-our-bodies-and-its-also-how-we-express-ourselves-play.html*, 3/26/2010.

[24] P. Richard-Amato. *Making it happen: Interaction in the second language classroom*. Longman, 1988.

[25] J. Roberts and R. Styron. Student satisfaction and persistence: Factors vital to student retention. *Research in Higher Education Journal*, 6(3):1–18, 2010.

[26] B. Rogoff. *Apprenticeship in thinking*. Oxford University Press, 1990.

[27] E. B. H. Sandseter and L. E. O. Kennair. Children’s risky play from an evolutionary perspective: The anti-phobic effects of thrilling experiences. *Evolutionary Psychology*, 9(2):257–284, 2011.

[28] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, Massachusetts, 1978.

[29] L. S. Vygotsky. Thinking and speech. In *The collected works of L. S. Vygotsky*, volume 1, pages 249–250. Plenum, 1987.

[30] L. S. Vygotsky. The problem of age. In *The collected works of L. S. Vygotsky*, volume 5, pages 187–205. Springer, 1998.

[31] S. A. Wallace, I. Russell, and A. Markov. Integrating games and machine learning in the undergraduate computer science classroom. In *GDCSE 2008*, pages 56–60, Miami, Florida, March 2008.

[32] L. Williams. Pair programming. In P. A. Laplante, editor, *Encyclopedia of Software Engineering*, volume II. Taylor and Francis Group, 2011.

[33] D. Xu, D. Blank, and D. Kumar. Games, robots, and robot games: Complementary contexts for introductory computing education. In *GDCSE 2008*, pages 66–70, Miami, Florida, March 2008.