# An impact comparison of two instructional scaffolding strategies employed in our programming laboratories:

## Employment of a supplemental teaching assistant versus employment of the pair programming methodology.

Ronald Erdei
Department of Mathematics and Computational Science
University of South Carolina Beaufort
Bluffton, SC 29909 United States

John A. Springer, David M. Whittinghill
Department of Computer and Information Technology
Purdue University
West Lafayette, IN 47906 United States

***Abstract*—Instructional scaffolding is a well-researched, commonly-practiced educational technique whereby support is temporarily provided as an individual learns [1]. Grounded in constructivist teaching and learning theory, scaffolding historically referring to support provided by a teacher to a student. However, in the modern learning environment, instructors have access to a wide range of tools and techniques with which to help them scaffold learners [2]. This paper reports the findings of a design experiment comparing the employment of two instructional scaffolding strategies in reducing the impact of large laboratory class size on undergraduate students enrolled in an introductory computer programming course having a mandatory laboratory component. The design experiment compared the performance of students on programming procedural knowledge assessments as well as programming self-beliefs in two offerings of the course, each treated as a cohort, which differed in the instructional scaffolding strategy employed in the mandatory laboratory component. In one cohort, an undergraduate teaching assistant was used as a supplement to the laboratory instructor while students completed programming exercises in the laboratory component of the course. In the other cohort, a cooperative programming methodology known as pair programming was used as a supplement to the laboratory instructor while students completed programming exercises in the laboratory component of the course. Cohort composition, including school of study (e.g., Liberal Arts, Science, Technology, Engineering, Business, etc.), student classification (i.e., freshman, sophomore, junior, senior, or other), gender ratio, and amount of prior programming experience, was comparable. Course content, administrative processes and assessment mechanisms (formative and summative) remained constant. Each student completed a summative assessment of programming procedural knowledge at weeks 7, 12, and 16 of the semester. Open ended feedback was solicited from each student via anonymous questionnaire during weeks 8 and 16 of the semester. Finally, each student completed a 19-item Likert-scale questionnaire investigating their self-beliefs on 5 constructs: debugging self-efficacy, programming self-concept, programming interest, programming anxiety, and programming aptitude mindset. The questionnaire employed to investigate self-beliefs is based on the Scott & Ghinea [3] instrument, modified for use in the specific context of the course. Our results indicate that the two scaffolding strategies provided comparable support to student learning during the first 12 weeks of the semester. However, during the last 4 weeks of the semester, the cohort scaffolded by a supplemental teaching assistant slightly outperformed the cohort scaffolded by employment of the pair programming methodology. This performance difference, though small, was statistically significant. Student programming self-beliefs, however, remained comparable between cohorts throughout the entire semester. Interestingly, the theme of 'immediacy of assistance' revealed itself during analysis of student open-ended feedback in both cohorts. These findings are considered using as lens Vygotsky's Zone of Proximal Development theoretical concept [4]. Implications are discussed for instructional practitioners considering employing these scaffoldings strategies in their own learning environments. (*Abstract*)**

***Keywords—instructional scaffolding; cooperative learning; computer programming pedagogy; pair programming; undergraduate teaching assistant; programming self-beliefs; (****key words****)***

## I. INTRODUCTION

Students often report that learning to program is challenging. Literature spanning the last two decades supports this anecdotal evidence [5, 6, 7, 8, 9], with two recent studies each finding that nearly one in three students enrolled in a computer programming course fails to complete the course [10, 11]. Motivated by a desire to assist our students in learning to program, we wondered: what interventions were available to improve the process by which students learn to program?

One means to improve the learning process is through appropriate use of instructional scaffolding. Instructional scaffolding is a well-researched, commonly-practiced educational technique whereby support is temporarily provided as an individual learns [1]. Originally used to describe the process by which an adult assists a child to learn [2], in the

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

modern computer programming classroom there are several instructional scaffolds available to instructors.

This study represents the third iteration in a series of design experiments investigating the use of pair programming, a collaborative computer programming methodology readily employed as an instructional scaffolding technique, to improve the computer programming learning process in a natural educational setting. Earlier iterations of study were smaller in scale and can be considered pilot studies for the current investigation. These earlier studies found that students employing the pair programming methodology in our programming courses regarded the experience positively, and that benefits and challenges common to collaborative learning strategies were both reported by students and observed by instructors [10]. Building upon these earlier findings, the current study investigates student performance on standard course assessment mechanisms, as well as student programming self-beliefs, between two offerings of an introductory computer programming course that differed in the instructional scaffolding strategy employed in the mandatory laboratory component of the course. In one offering, the pair programming methodology is employed as the instructional scaffolding used to supplement the laboratory instructor in the mandatory computer programming laboratory. In another offering, an undergraduate teaching assistant is employed as the instructional scaffolding used to supplement the laboratory instructor in the mandatory computer programming laboratory.

## II. Background

### A. Instructional Scaffolding

The term scaffolding was coined by Wood, Brunner, and Ross in their 1976 investigation into the role of tutoring on the development of problem solving abilities in children [13]. In this study, Wood, Brunner, and Ross held that the scaffolding process "enables a child or novice to solve a problem, carry out a task or achieve a goal which would be beyond his unassisted efforts". They focused on the actions of an expert, typically an adult, who was responsible for "controlling those elements of the task that are initially beyond the learner's capacity" thus allowing the learner to complete only the elements of the task that are within their ability. The modern view of instructional scaffolding reflects a perspective far broader than simply that of expert tutor, holding instructional techniques such as teacher-modeling, tools such as prompt-providing software, and even peers with whom the learner collaborates as instructional scaffolds [2].

Fundamental to instructional scaffolding is the Zone of Proximal Development (ZPD), a theoretical construct described by Vygotsky [4]. The ZPD refers to a conceptual zone of tasks challenging enough that a learner is unable to complete them alone, but is able to complete them with assistance. It is often visualized in the form of a Venn diagram, as depicted in Fig. 1. Notice that Fig. 1 depicts three clearly delineated areas: the area containing tasks that the learner can perform without assistance (i.e., learners can do these things on their own), the area containing tasks that the learner can perform with assistance, but which otherwise would be unachievable (i.e., the ZPD), and the area containing tasks that the learner is unable to perform even with assistance. Scaffolding theory posits that tasks to be learned should initially be in the ZPD of the learner. Assistance is temporarily provided, allowing the learner to successfully perform these tasks; and through performing these tasks, even with assistance, the learner develops the skills necessary to subsequently perform them on their own. Conceptually, the tasks move inward from the ZPD into the set of tasks learners can perform without assistance. Assistance, no longer being necessary for the learner to perform the given task, is then removed. This gradual reduction and eventual elimination of learner assistance is known as fading, and is a distinguishing characteristic of instructional scaffolding [14].

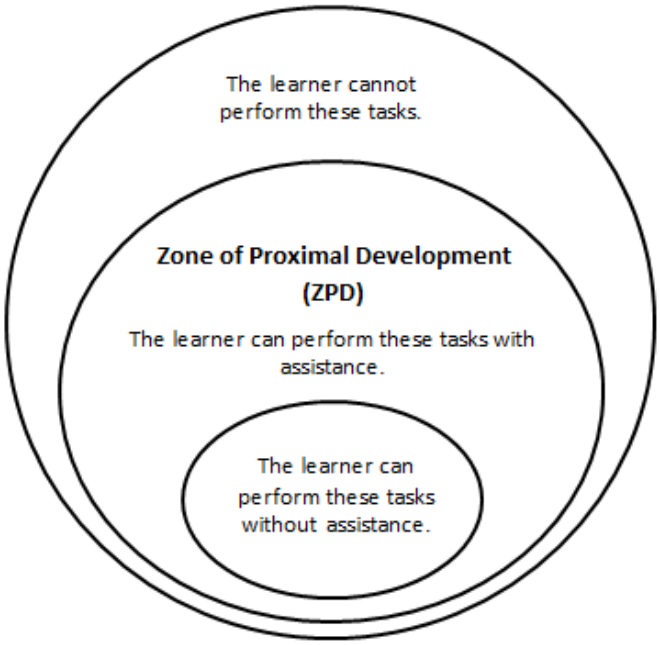


Fig. 1. Zone of Proximal Development

### B. Cooperative Learning

Cooperative learning is the "instructional use of small groups" to enable students working together to "maximize their own and each other's learning" [15]. One of the primary mechanisms by which learning occurs when working cooperatively is through group member conversations. The processes of externalizing and articulating developing knowledge facilitates learning [16], with the most effective learning occurring when learners articulate their still unformed understanding [1].

### C. Pair Programming

As depicted in Fig. 2, pair programming is a cooperative computer programming methodology in which two individuals, literally working side by side on a single computer, assume complimentary roles in the active pursuit of a programmatic solution [17]. One role, that of the driver, controls the keyboard, typing the code (or creating the document) while focusing on the details of the program. The second role, that of the navigator, "actively observes the work of the driver, looking for tactical and strategic defects, thinking of

alternatives, writing down 'things-to-do', and looking up references" [18]. The two individuals periodically switch roles, repeatedly iterating between driver and navigator, so that each individual spends equal time acting in each of the two roles.

Fig. 2. Students employing the Pair Programming Methodology

## III. METHODOLOGY

The investigation compared the performance of students on programming procedural knowledge assessments as well as programming self-beliefs in two offerings of the course, each treated as a cohort, which differed in the instructional scaffolding strategy employed in the mandatory laboratory component. In one cohort, an undergraduate teaching assistant was used as a supplement to the laboratory instructor while students completed programming exercises in the laboratory component of the course. In the other cohort, a cooperative programming methodology known as pair programming was used as a supplement to the laboratory instructor while students completed programming exercises in the laboratory component of the course. Cohort composition, including school of study (e.g., Liberal Arts, Science, Technology, Engineering, Business, etc.), student classification (i.e., freshman, sophomore, junior, senior, or other), gender ratio, and amount of prior programming experience, was comparable. Course content, administrative processes and assessment mechanisms (formative and summative) remained constant.

### A. Learning Environment

The investigation was conducted in the natural educational setting of an introductory computer programming course offered at a large Midwestern University. The course was sixteen weeks in duration, had no prerequisite courses, required no prior programming experience, and was considered a service course. Students enrolled in the course were from non-computing and non-IT disciplines, were overwhelmingly upper-class undergraduates, and reported little if any prior programming experience. The mandatory lecture component of the course met twice per week, each meeting being 50 minutes in duration, for the full sixteen weeks of the semester. The lecture component of the course focused students on the learning of computer programming conceptual, or declarative, knowledge. The mandatory laboratory component of the course met once per week, each meeting being 110 minutes, for the full sixteen weeks of the course. The laboratory component of the course focused students on the learning of computer programming procedural knowledge (i.e., "how to" create computer programs); to this end, it employed performance tasks as the primary instructional strategy. A performance task is a learning activity in which learners perform action sequences and procedures to demonstrate their procedural knowledge, often resulting in a tangible product or physical performance [19]. During the weekly laboratory component of the course, students completed performance tasks, typically the creation of an appropriately challenging computer program, with instructional scaffolding providing by a laboratory instructor.

### B. Treatment

The investigation was conducted over two consecutive semesters of the course. Course content, administrative processes and assessment mechanisms (formative and summative) remained constant, while the instructional scaffolding method supplementing the laboratory instructor in the mandatory laboratory component of the course differed. In one cohort (n=89), an undergraduate teaching assistant was used as a supplement to the scaffolding provided by the laboratory instructor. In the other cohort (n=110), the pair programming methodology was used as a supplement to the scaffolding provided by the laboratory instructor. The structure of the course laboratory, including scaffolding strategy, is shown in Fig. 3.

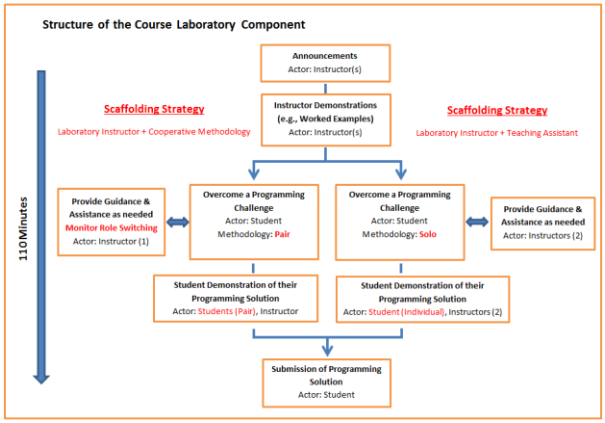


Fig. 3. Comparison of Laboratory Structure for the Two Scaffolding Strategies

### C. Data Collection

Each student completed a summative assessment of programming procedural knowledge at weeks 7, 12, and 16 of the semester. Each summative assessment took the form of a timed, observed examination in which students were required to implement working computer programs based on instructor provided specifications. Each examination was conducted in the student's regularly scheduled computer laboratory, under direct supervision of the laboratory instructor, and was completed within one standard computer laboratory period (i.e., students had a maximum of 110 minutes to complete each examination). Open ended feedback was solicited from each student via anonymous questionnaire during weeks 8 and 16 of the semester. Finally, each student completed a 19-item Likert-scale questionnaire investigating their self-beliefs on 5 constructs: debugging self-efficacy (DSE), programming self-

concept (PSC), programming interest (INT), programming anxiety (ANX), and programming aptitude mindset (APT). The questionnaire employed to investigate self-beliefs is based on the Scott and Ghinea instrument [3], modified for use in the specific context of the course.

TABLE I. QUESTIONNAIRE QUESTIONS

| ID | Text |
|---|---|
| DSE1 | I am confident that I can understand Visual Basic exceptions (e.g., Format Exception). |
| DSE2 | I am confident I can solve simple problems with my programs. |
| DSE3 | I am confident I can implement a method from a description of a problem or algorithm. |
| DSE4 | I am confident I can debug a program that calculates prime numbers. |
| PSC1 | I am just not good at programming. |
| PSC2 | I learn programming quickly. |
| PSC3 | I have always believed that programming is one of my best subjects. |
| PSC4 | In my programming labs, I can solve even the most challenging problems. |
| INT1 | I enjoy reading about programming. |
| INT2 | I do programming because I enjoy it. |
| INT3 | I am interested in the things I learn in programming classes. |
| INT4 | I think programming is interesting. |
| ANX1 | I often worry that it will be difficult for me to complete debugging exercises. |
| ANX2 | I often get tense when I have to debug a program. |
| ANX3 | I get nervous when trying to solve programming bugs. |
| ANX4 | I feel helpless when trying to solve programming bugs. |
| APT1 | I have a fixed level of programming aptitude, and not much can be done to change it. |
| APT2 | I can learn new things about software development, but I cannot change my basic aptitude for programming. |
| APT3 | To be honest, I do not think I can really change my aptitude for programming. |

## IV. RESULTS

Our results indicate that the student cohort that employed the cooperative pair programming methodology as a supplemental instructional scaffold performed slightly better on programming examination 1 and programming examination 2 than the student cohort that employed the undergraduate teaching assistant as a supplemental instructional scaffold. However, this difference was not statistically significant. In contrast, the student cohort that employed the cooperative pair programming methodology as a supplemental instructional scaffold performed slightly worse on programming examination 3 than the student cohort that employed the undergraduate teaching assistant as a supplemental instructional scaffold; this difference was statistically significant. These results are shown in Fig. 4.

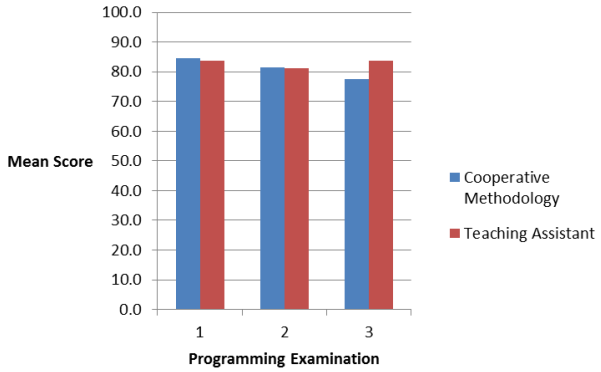


Fig. 4. Student Performance on Course Programming Examinations by Scaffolding Strategy

Our results also indicate that, regardless of scaffolding strategy employed in the course laboratory, the student cohorts displayed no difference in programming self-beliefs at the conclusion of the course. These results are shown in Fig. 5.

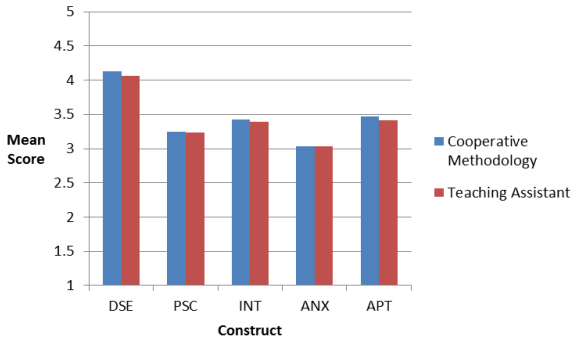


Fig. 5. Student Self-Beliefs by Scaffolding Strategy

In analyzing student feedback, an interesting theme emerged: *immediacy of assistance*. Two examples of feedback displaying this theme are:

- "Working with a partner inlab assignment is very helpful because it helps engage in conversation with your partner and in the same time you get feedback while you are trying to solve a problem"
- "Having a partner in the lab makes sure that I don't get stuck on anything and end up having to figure anything out or wait for a teacher."

## V. CONCLUSIONS

Our results indicate that the two scaffolding strategies provided comparable support to student learning during the first 12 weeks of the semester. However, during the last 4 weeks of the semester, the cohort scaffolded by a supplemental teaching assistant slightly outperformed the cohort scaffolded by employment of the pair programming methodology. This performance difference, though small, was statistically significant. Student programming self-beliefs, however, remained comparable between cohorts throughout the entire semester. Interestingly, the theme of 'immediacy of assistance'

revealed itself during analysis of student open-ended feedback in both cohorts.

## VI. DISCUSSION

Both scaffolding strategies enabled comparable student learning during the early part of the semester; that is, while the course procedural knowledge was less challenging. However, as the course procedural knowledge became more challenging, the supplemental instructional scaffolding provided by the undergraduate teaching assistant enabled better procedural learning when compared to that provided by the cooperative programming methodology. We believe this reflects the level of task difficulty exceeded the ZPD, forcing students to rely more heavily on the instructional scaffolding provided by the laboratory instructor and undergraduate teaching assistant. As the student cohort employing the cooperative programming methodology lacks an undergraduate teaching assistant, it must rely solely on the laboratory instructor for assistance. We believe a modified Zone of Proximal Development diagram, one with 2 distinct ZPDs as shown in Fig. 6, better reflects the scaffolding available when students pair program.

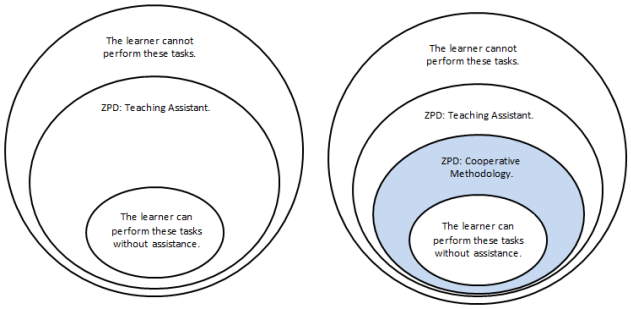


Fig. 6. ZPD Diagram with 1 Scaffold vs. Diagram with 2 Scaffolds

Interestingly, student feedback did not emphasize the fact that assistance was necessary (i.e., the work was too difficult) but instead the immediacy of available assistance (i.e., wait time for an instructor). Reflecting this, student feedback from the cohort employing the cooperative programming methodology emphasized the detrimental effects of having only a single instructor in the laboratory, specifically identifying the high level of competition for instructor assistance, as well as the lengthy delays in receiving instructor assistance, once the level of difficulty exceeded the ZPD of partner scaffolding.

As Masters and Yelland point out, instructional scaffolding "needs to be modified to suit the circumstances of implementation" [20]; the scope of the task and the learner's own ZPD need to be taken into account if instructional scaffolding is to be successful in assisting learners. Our findings indicate that, should we wish to continue employing the pair programming methodology as instructional scaffolding, our students would benefit in the latter part of the semester by modifications to our current instructional strategy, current scaffolding, or current learning environment. The most direct means of changing the circumstances would be to complement pair programming with a supplemental instructor in the classroom during weeks 13, 14, and 15; in essence, strategically employing a second instructor only when collaborative scaffolding alone is insufficient support for student learning. An alternate means of changing the circumstances would be to adjust the location of the performance task relative to the student's current ZPD as depicted in Fig. 7.

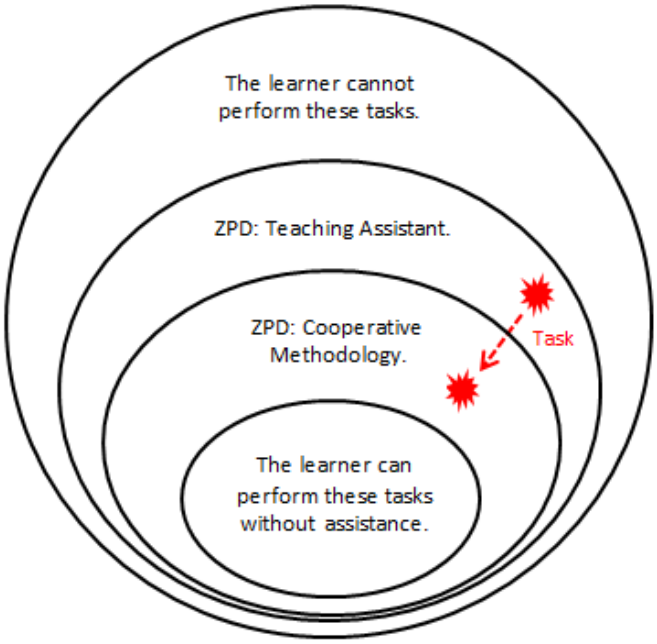


Fig. 7. Changing Task Scope relative to Learner's ZPDs

Instructors have at their disposal a great number of tools by which to catalyze the change in task location relative to student ZPD. For example:

- Students could watch videos illustrating appropriate implementation techniques and strategies prior to the laboratory.
- Students could complete targeted readings prior to the laboratory.
- Students could diagram, in part or in whole, the computer program's control/logic flow prior to the laboratory.
- Students could implement, in part or in whole, a similar computer program under instructor-led demonstration.

## REFERENCES


[1] R.K. Sawyer. *The Cambridge Handbook of the Learning Sciences*. New York: Cambridge University Press, 2006, pp. 11-12.

[2] B. Rosenshine, and C. Meister. "The Use of Scaffolds for Teaching Higher-Level Cognitive Strategies." Educational Leadership, vol. 49(7), pp. 26–33, 1992.

[3] M. Scott, and G. Ghinea. (2014). "Measuring Enrichment: The Assembly and Validation of an Instrument to Assess Student Self-Beliefs in CS1" in *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 2014, pp. 123–130.

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

[5] T. Jenkins. “On the Difficulty of Learning to Program” in *Proceedings for the 3rd annual Conference of the LTSN Centre for Information and Computer Sciences*, 2002, pp. 53-58.

[6] B. Hanks, C. McDowell, D. Draper and M. Krnjajic. “Program quality with pair programming in CS1” in *Proceedings of the 9th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2004, pp. 176-180.

[7] A. Robins, J. Rountree and N. Rountree. “Learning and Teaching Programming: A Review and Discussion.” *Journal of Computer Science Education*, vol. 13(2), pp. 137-172, 2003.

[8] T. Boyle, C. Bradley, P. Chalk, R. Jones, & P. Pickard. “Using Blended Learning to Improve Student Success Rates in Learning to Program.” *Journal of Educational Media*, vol. 28(2–3), pp. 165–178, 2003.

[9] S. Bergin and R. Reilly. “The influence of motivation and comfort-level on learning to program” in *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group*, 2005, pp. 293-304.

[10] J. Bennedsen and M. Caspersen. "Failure rates in introductory programming" in *ACM SIGCSE Bulletin*, vol. 39(2), pp. 32–36, 2007.

[11] C. Watson and F. Li. (2014). “Failure rates in introductory programming revisited” in *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, 2014, pp. 39–44.

[12] R. Erdei, D. Whittinghill, and J. Springer. (2014). “Collaboration While Programming: Observing Student Perceptions of Pair Programming in the Classroom” in *Proceedings of the World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, 2014, pp. 543–551.

[13] D. Wood, J.S. Bruner, and G. Ross. “The Role of Tutoring in Problem Solving.” *Journal of Child Psychology and Psychiatry*, vol. 17(2), pp. 89–100, 1976.

[14] P. Sharma, and M. Hannafin. “Scaffolding in technology-enhanced learning environments” in Interactive Learning Environments, vol. 15(1), 27–46, 2007.

[15] D. Johnson, and R. Johnson. “Cooperation and the Use of Technology” in *Handbook of Research on Educational Communications and Technology*, 3rd ed. J. M. Spector, Ed. New York: Lawrence Erlbaum Associates, 2008, pp. 785–811.

[16] J. Bransford, A. Brown, and R. Cocking. (2000). *How People Learn: Brain, Mind, Experience, and School*. Washington, D.C.: National Academy Press, 2000.

[17] ] K. Beck, and C. Andres. *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley, 2004, pp. 42-43.

[18] B. Arisholm, H. Gallis, T. Dyba, and D. Sjoberg. “Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise”. IEEE Transactions on Software Engineering, vol. 33(2), pp. 65–86, 2007.

[19] T. Kubiszyn, and G. Borich. *Educational Testing and Measurement: Classroom Application and Practice*. Glenview: John Wiley & Sons, Inc, 2004, pp. 186-188 .

[20] N. Yelland, and J. Masters. “Rethinking Scaffolding in the Information Age”. *Computers and Education*, vol. 48(3), 362–382, 2007.