Teaching for Understanding in Computer Science Education

William A. Booth, Ph. D.
Bill_Booth@baylor.edu
Baylor University, Waco, Texas 76798

Abstract

The Teaching for Understanding (TfU) provides a framework for teaching that is focused on student understanding. In computer science education, pair programming is a powerful tool for helping students develop sound problem solving, and programming skills. This article discusses how pair programming can be used in the Teaching for Understanding framework.

Introduction

The teaching of computer programming requires the instructor to facilitate the learning of those skills that will allow the students to be successful in the world of professional software development. Unfortunately, important skills used in professional programming are not commonly used in teaching introductory computer programming students (Williams, 2002). In industry, programmers collaborate for the majority of their day. Yet, most often while completing their degree, programmers must learn to program alone; collaboration is considered cheating. Communication skills and interaction skills are critical to the collaborative development process found in the software development industry. Much of the learning in higher education falls short of the threshold experience needed for deep understanding; students learn the pieces but never have an opportunity to put the pieces together (Perkins, 2009, p. 9). Computer science educators can learn both from the methods used in commercial program development and from research on how students learn. Specifically, pair programming is a powerful software development methodology that resonates with the principles and theoretical underpinnings of the teaching for understanding framework (TfU).

Teaching for Understanding Framework

The four parts of the TfU framework are the generative topic, understanding goals, performance of understanding, and ongoing assessment (Wiske, 1998). The generative topic defines what is worth understanding. It should be of interest to the learners as well as the instructor. Software development is the generative topic in computer science education. The number of computer science majors is on the rise (Duffy, 2009) which is an indirect indicator that students find the field interesting. For computer science educators and students the idea of developing software is generative. The second part of the TfU framework is the understanding goals. Understanding goals are the fundamental ideas in the discipline (Wiske, 1998). These goals serve to inform both the teacher and the learner about the most important outcomes from learning about the generative topic. Typical understanding goals in computer science are to demonstrate the ability to design, document, implement and test software applications. The third part of the TfU framework is the performance of understanding. How will the learner demonstrate that they understand the content taught about the generative topic? Again, in computer science the answer is the software application. Can the learners design, document, implement and test a software application that is correct, reusable and scalable? This type of performance of understanding allows students to develop and demonstrate their understanding of the generative topic. The final part of the TfU framework is ongoing assessment. The ongoing assessments should be frequent and based on well known criteria and derived from the understanding goals.

In addition to addressing the four parts of the TfU framework the learner's prior knowledge must be considered. "Students come to the classroom with prior knowledge that must be addressed if teaching is to be effective" (Darling-Hammond, 2008, p. 3). TfU assumes a constructivist

epistemology which implies that knowledge is constructed not transmitted. The construction of new knowledge must be built on the foundation of what is already known. An example of how prior knowledge interferes with learning in computer science is the misconception about algorithm efficiency (Gal-Ezer, 2004). Students commonly believe that the code length is an indication of the algorithm efficiency. This misconception leads them to believe that shorter code segments always run faster. Even after formal instruction about algorithm efficiency students have difficulty disregarding code length when determining how an algorithm will perform. Teachers must also help learners organize their new knowledge in a conceptual way that will allow them to apply this knowledge outside of the classroom (Darling-Hammond, 2008). The ability to apply knowledge learned in one context to a different context is known as transfer (Bransford, 2000). This is a key characteristic of experts. Experts have the ability to recognize similar problems even when encountered in different contexts. Teachers must help student to contextualize their knowledge in a way that promotes transfer (Darling-Hammond).

A third way teachers can help promote student understanding is metacognition. "Students learn more effectively if they understand how they learn and how to manage their own learning (Darling-Hammond, 2008). This suggests that teachers should regularly ask the students to reflect on what they know and how they came to understand these concepts. Teachers should also help learners to identify the practices that help them learn and reflect on the methods that promote their personal understanding.

These three principles, the impact of prior knowledge, conceptualizing knowledge, and metacognition impact the application of the TfU. Teachers must work to identify generative topics that are meaningful and have practical application in the field (Darling-Hammond, 2008). They must allow the learners to actively pursue the acquisition of new knowledge and

understanding; pair programming is particularly well suited for this task. Finally effective teaching must ground learning in the prior knowledge of the learner, correcting misconceptions when they are identified.

Pair Programming

Pair programming is a technique used in the software development process known as Extreme Programming (XP). XP was developed by Kent Berk in the late 1990's and first published in his book "Extreme Programming Explained" in 1999. This style of programming requires two programmers to work together on one computer while solving the same problem. The member of the team at the computer is called the driver and is responsible for typing the program. The other member of the team, called the navigator, provides support for the driver by evaluating the program as it is developed (Williams, 2002). McDowell describes pair programming as:

… teams of two programmers working simultaneously on the same design, algorithm, code, or test. Sitting shoulder to shoulder at one computer, one member of the pair is the "designated driver," actively creating code and controlling the keyboard and mouse. The 'non-driver' constantly reviews the keyed data in order to identify tactical and strategic deficiencies, including erroneous syntax and logic, misspellings, and implementations that don't map to the design. After a designated period of time, the partners reverse roles. (McDowell, 2006)

This methodology for software development facilitates the ongoing assessment that is critical to the TfU framework.

Jehng (1997) showed that this type of learning requires the two team members to achieve shared understanding of each action taken in a particular situation. This convergence of two different

states of knowledge is an interactive process of displaying, confirming, rejecting, and modifying meaning co-constructed by the learners. When students work together they accept more responsibility for their own learning (Lippert, 1997). Students working in groups ask fewer questions requiring less of the instructors time, because they tend to attempt to solve the problems independently before asking for help (Williams, 2002). This research clearly supports the pair programming methodology of software development. A critical element of pair programming is evaluation. Gnagey (1997) showed that there is a substantial positive correlation between self-evaluation and the evaluation given from other team members. Students seem to be able to accurately evaluate their own work and the work of others on the team. Gnagey (1997) also showed a positive correlation between students' self-evaluation and the formal grades earned in their course work. It seems that students are capable of accurately evaluating their own work and the work of others, both on the same team and on other teams. This finding suggests that students will be able to self regulate the effectiveness of their pair-programming team. It would also seem reasonable to assume that because students have the ability to monitor the quality of their teams work, they are probably the best judges of when to ask for additional help from the instructor. In support of the positive effect of peer-evaluation, Gehringer (2001) found that in three different courses, in which peer-evaluation was used, the students reported that the peer-evaluation was helpful. The next section will look at how pair programming can be integrated with the TfU framework.

Computer Science Education

A powerful and effective model of computer science education can be created by combining the TfU framework and pair programming. First there is the issue of the generative topic. Problem

solving through the creation of software applications in computer science is naturally generative. All problems can be solved in multiple ways and there is seldom one right answer. The key to making this problem solving generative is to embed it in a collaborative environment. The educator must identify an aspect of computer science that will be engaging for both the learner and the teacher. Some students feel that learning how to program computers is difficult and tedious (Stephenson, 2002). Computer science is often perceived as machine-focused and isolating. When many students think about computing, they imagine people spending all their time sitting at a computer. Working in groups will help the students to become more engaged in the learning process and help to motivate the student to learn the more complex concepts (Lippert 1997). Self directed teams, such as those used in pair-programming, have been shown to produce positive results in learning computer skills (Buffington 1998). The goal of software development is fundamentally problem solving and this study shows that pair programming helps to increase solution quality and the learners confidence in their ability to solve problems. The performance of understanding in computer science is the software application created by the learners. Ultimately the goal is to assist the learner in transitioning from a novice to an expert programmer/problem solver. As this transition begins the novice will begin to organize the computer science knowledge, increasing their capacity for pattern recognition and information chunking (Bransford, 2000). The learners should also gain fluency in information retrieval. Initially the learners may need to refer to notes and manuals to retrieve commonly used program syntax. Over time this behavior should decline. In addition to behaving more like experts, the introductory programming student will demonstrate their understanding of problem solving through software development by designing, documenting, implementing and testing software applications.

Perhaps the greatest strength of pair programming is that it requires the learners to engage in continuous evaluation of their work. As the designated driver creates the code their partner is evaluating the quality of the work. McDowell (2006) demonstrated that pair programming helps with retention, program quality and programmer confidence. These beneficial effects are true for both men and women. The learners will benefit by sharing their understanding of the software development process and the particular problem being solved.

The demonstration of understanding is the software application. Although the source code is sufficient evidence to demonstrate understanding to an expert, code review is an additional method for evaluating the quality of software. Formal code inspections are used in industry to identify software defects and improve quality (Hundhausen, 2009). After reviewing a piece of code, members of the review team discuss the strengths and weaknesses of the applications source code. Hundhausen (2009) found that when code reviews were used in an introductory programming course, the quality of the code improves, there was an increase in the frequency and quality of discussion related to software development and best practices, and it contributed to an overall sense of community in the classroom.

An additional aspect of the TfU framework is the progression of performance. As learners engage in the learning process, embedded in the TfU framework, they typically progress from messing about, to guided inquiry and finally the culminating performance (Wiske, 1998). Again pair programming is well suited for this progression. In the early part of the course learners need time to familiarize themselves with the development environment and syntax of the programming language. Messing about is ideal to accomplish this task. Quickly this should transition into guided practice, where the instructor introduces specific language features and computational theory. Guided practice can then be used to help the students master these

constructs. Finally, with little interaction with the instructor, the learners should create a culminating performance to demonstrate their understanding of the programming language, development environment, and the software development process.

Summary

The use of pair-programming has been demonstrated to reduce the number of errors in student code, reduce the amount of time teachers spend answering questions and improve the overall quality of the programs (Williams, 2002). This finding seems to fit well with what might be intuitively expected. When two individuals work together the end result is improved. McDowell's (2006) work suggests that the fear that one partner will do all the work and the other partner will not learn effectively is not a typical scenario. When pair programming is viewed through the TfU lens all the key components of the framework are present. Software development is the generative topic, the ability to design, document, implement and test software is the understanding goal. A potential concern about using the TfU is the need for ongoing assessment. This concern is motivated from a perspective that requires the instructor to perform each step of the assessment. Using pair programming will shift the burden of ongoing assessment from the instructor to the students. With the burden of assessment distributed to all pairs, the potential for successfully using the TfU framework is dramatically increased. Finally the software created by the learners becomes the demonstration of understanding. When the code is read by an expert, it can reveal how the student understands of the problem being solved and their understanding of the software development process. Software is a creative work, much like an essay written in an English course, with many subtle nuances. Well written code requires a deep understanding of the problem being solved and the software development process.

Interesting problems being solved using the pair programming strategy provides an ideal environment for the application of the TfU framework to the teaching of computer science.

References

Bransford, J. D., Brown, A. L., and Cocking, R. (2000). *How People Learn*, National Academy

Buffington, J. R. (1998). *Self-Directed Teams in the Introductory Information Systems Course: Lessons Learned* [EDRS No. ED 431 420].

Darling-Hammond, Linda (2008). *Powerful Learning: What we know about teaching for understanding*. Jossey-Bass Publishers: San Francisco.

Duffy, C. (2009). *Computer science major is cool again*, Network World ,March 17, 2009, from http://www.networkworld.com/news/2009/031409-computer-science-majors.html

Gal-Ezer, Zur. E. (2004), *The efficiency of algorithms - misconceptions*. Computers and Education, 42(3),

Gehringer, E. (2001). (Electronic Peer Review and Peer Grading in Computer-Science Courses). *2001 SIGCSE Conference on Computer Science Education*.

Gnagey, W. J., Sarles, R. B., & Sarver, T. R. (1997). *Correlates of Self Concept in Collaborative Learning* [EDRS No. ED 413 812].

Hundhausen , C., Agrawal A., Fairbrother D., Trevisan M. (2009) *Integrating Pedagogical Code Reviews into a CS 1 Course: An Empirical Study,* SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.

Jehng, J.-C. J. (1997). *The Psycho-Social Processes and Cognitive Effects of Peer-based Collaborative Interactions with Computers*. Journal of Educational Computing Research, *17*(1), 19-46.

Lippert, S. K., & Granger, M. J.. *Peer Learning in an Introductory Programming Course*. In 12th Annual Conference of the International Academy for Information Management.

McDowell, C., Werner, L., Bullock, H., Fernald, J. (2006). *Pair Programming Improves Student Retention, Confidence, and Program Quality*, Communications of the ACM, 49(8), 90–95.

Perkins, David (2009). *Making Learning Whole: How Seven Principles of Teaching Can Transform Education*. Jossey-Bass Publishers: San Francisco Press,Washington, DC.

Stephenson, C. (2002). *Computer Science Education: Looking Back and Looking Ahead*. Learning and Leading with Technology, *30*(2), 6-9, 44-5.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). *In Support of Pair Programming in the Introductory Computer Science Course*. Computer Science Education.

Wiske, Martha Stone, Editor (1998). *Teaching for Understanding*. Jossey-Bass Publishers: San Francisco.