PEER-LED TEAM LEARNING IMPLEMENTATION

AN ARGUMENT: WHY NOT HAVE PEER-LED TEAM LEARNING IN COMPUTER SCIENCE?

IGOR LABUTOV

An effective and innovative approach to a classical classroom is ironically a natural extension of everyday learning. Peer-Led Team-Learning (PLTL), an educational model developed by Dr. David Gosser at the City College of New York, is an interactive learning atmosphere that emphasizes team-based collective learning, led by fellow students who have successfully completed the given course. The primary goal of PLTL, however, is to decentralize the classical approach, through eliminating one-way communication from the instructor to the students. The foundation of the novel approach is the looped feedback system, through which the leader and the students raise their development level autonomously. The approach proved to be highly successful, and is part of the General Chemistry course at the City College of New York (CCNY).

Because the entire principle behind peer-led learning is collaboration and collective participation, its application in only one course isolates and severely limits its potential for other courses. In particular, almost every new Peer Leader immediately notices students' deficiencies in mathematical and logical thinking. General Chemistry is a highly computational course that involves concrete problem-solving skills and analytical approaches. These skills apply especially to Mathematics and Computer Science. These are so-called "bridge" courses, through which the students lay their foundation for higher-level courses in sciences. Among the numerous courses which require immediate attention, CSc102, Introduction to Computing, stands out as the gap that keeps so many of the students from excelling in their fields of interest. It is important to consider this as a crucial steppingstone on their path to higher-level sciences and engineering courses.

CSc102 has one of the highest drop-out rates of all courses offered at CCNY. It also has one of the strangest grade distributions. For example, after the midterm, which was worth 45 points, more than two thirds of the class received below 20 and three students achieved an almost perfect score. Almost no one averaged between the two extremes, signifying a very strong deviation from the norm. The difference is quite apparent, and lies in the fact that the students who did very well already knew the material, and their grade reflected knowledge and practice attained elsewhere. Having already done programming, perhaps even in another language, the student is prepared for the classical style of lecture to teaching programming. This includes both the theoretical foundation in abstract thinking, problem-solving and most importantly critical thinking in being able to break apart a problem of any complexity to a set of distinct and simple modules.

This leads to an important observation, that CSc102 aims at performing two tasks at the same time – introducing students to the unfamiliar science of computing, while plowing through the syntax and detailed structure of an arbitrarily selected language. This burdens the students not only to struggle in understanding the cryptic, unfamiliar syntax, but at the same time fitting it in within a much larger scope of solving a given problem. The consequence is obvious and undeniable – students do not know where to begin. One visit to a

CSc102 lab will show a familiar scene of about fifteen students looking blankly at a computer screen filled with a few lines of code and a blinking cursor. Ironically, it is the code they wrote they struggle to understand. This is common. Given a problem in the lab, everyone immediately jumps to the keyboard, initializing variables of generic names and types (everyone will have their *i, j, x* and *y*) after which they may even include a header for a loop or a conditional statement, and it is then where their problems begin. In fact, their problem is most likely already finished. They have chosen a wrong path, but not because they wanted a shortcut, they were simply never taught otherwise.

This is where, through my experience serving as a Peer Leader, a PLTL program for CSc102 will immediately and without doubt result in a complete turn-around of how students see and perceive not only a single language, but an entire landscape of problem-solving in the context of the powerful tools available to them. The questions they must be able to answer is how these cryptic lines of code connect to a given problem at hand in whatever field of study they are pursuing. Curiosity and questions which students possess, and were unable to satisfy in lecture, would now become wide-open doors to an unfathomable land of exploration and learning, where the computer turns into an indispensable tool along their road of travel.

Peer-led groups of about eight people, in a comfortable and friendly atmosphere, without the presence of a professor, would most definitely uncover what has been missing in computing instruction. A peer-led group will be open to discussion, surveying various methods and approaches, and most importantly will concentrate, not on particular syntax, but on problem-solving techniques. Ask practically any CSc102 student, and they will tell you everything about each conditional statement, each loop, where the brackets go and how functions are declared. Now give them a real-world problem and ask to write a program to solve it. They might know all the modules that make up the program, but it is the same as trying to fix a radio with all the tools at hand, yet no idea where to begin.

Having workshop in CSc102 would be tremendously advantageous for this course. Even though syntax is highly rigid, approaches to solving the problem are literally unlimited. Each student in the workshop could propose a unique method, and most importantly, each one will be able to demonstrate and discuss it with everyone else. It is after this stage that the turning point in their understanding will take place. Now, they will question each line of code, analyze every function and every approach, and most importantly develop a mindset of a true scientist. They will search for efficiency, and for the first time will have a pool of goal-oriented classmates in offering their own views and perspectives on solving a problem. The Peer Leader will not be responsible for teaching the syntax or even offering ways of solving a problem, but merely serving as the motivating guide in the intellectual collaboration. It will be during these workshop hours that students will develop unrivaled analytical, problem-solving skills, which will lead them through all of their science courses and perhaps their career.

As a more concrete example, students will be initially introduced to the keystone of problem-solving techniques – flowcharts and logic diagrams. A problem of any complexity and depth can be broken up into small and manageable modules that are separated by conditional statements. They will be shown that it is absolutely required to construct a complete flowchart for a problem of any level of complexity, before touching a single key. This is why I suggest that it is important that the workshop not take place in a computer lab, but rather would allow the leader possession of a projector and a laptop running the compiler. This way, when students have successfully developed their plan of action, the strategy they will use to solve the problem, they will be able to come up and type up the module they will contribute, without trying out random code just to get the job done. Performance analysis will then follow as a discussion and if the code did not work, proper debugging can be done with each student proposing their way of finding an error. This way, the thought process will be audible to everyone, illustrating an indispensable tool of a scientist and engineer, collaborative brainstorming and team work. Most importantly, students will now possess enough information to have direction no matter how difficult or complex a given problem may appear. At any step of their programming, they will possess a tool to put them on a right path in finding a way to their goal.

My basis for emphasizing the development of a logic diagram or a flowchart is my own workshop experience in General Chemistry (first semester). It is fascinating to what extent the problems students are having in Chemistry coincide with those in Computing. Now, replace the computer lab with a chemistry classroom. Every student will describe atoms and molecules, laws that govern them, even the various mathematical relationships, such as between Pressure and Volume of a gas. Now, give them a three-step problem, and uniformly blank faces appear in every corner of the room. Situations like this arise with frightening regularity in my workshop, and from my conversations with other leaders, their situation is not much different. This results in a peculiar stalemate, where lecture professors are limited by time in only presenting material, and a workshop leader in practicing related problems, the students are not able to perform to standards, again because they cannot cross this unfortunate gap. There is barely enough time to cover new material, let alone discuss proper problem-solving approaches. I remember clearly drawing a generalized path of steps towards the solution before even looking at the numbers, let alone performing calculations. It is a routine experience, to assign a problem in Chemistry, and before even finishing a sentence, a few people are already arguing whether the two numbers should be divided or multiplied. This is a direct analogy to how CSc102 students leap to their keyboard, punching in variables they don't know if they even need. And this situation is not only observed in Chemistry, in fact, it is even worse in courses like Physics and higher-level Engineering classes, where material becomes so dense, and time limited, that students truly reach a barrier, solely due to their lack of problem-solving training. At the end, this all comes down to being able to compute – state your problem, draw up a plan, draw a diagram, and only then mechanically perform the steps. CSc102, in my opinion, is the perfect course in which to organize the workshop, as it will not only juxtapose the problem-solving techniques in the context of coding, but the benefit will echo throughout the Science and Engineering curriculum at CCNY.

Workshops in CSc102 will deliver far more than what is possible with a lecture and a lab. It will truly become a closed-loop feedback machine, engraving problem-solving approaches and collaborative brainstorming techniques critical in every area of science and engineering. Most importantly, the addition of the Workshop to the lecture and lab will complete the three crucial pieces required for great achievement in the course. Workshop will develop invaluable problem-solving techniques that will bridge the gap between abstract representations and real-world problems, the lecture will provide a tool, the proper coding and syntax to apply the learned techniques within the context of an autonomous group, and the lab will develop students' skills in individual work, where they will apply both, their knowledge of the language and problem-solving skills in a real-world, job environment.

Igor Labutov
Peer Leader and Computer Engineering Major
The City College of New York, CUNY

Cite This Article as: Labutov, I. (2012). An argument: Why not have Peer-Led Team Learning in Computer Science? Peer-Led Team Learning: Implementation. Online at http://www.pltlis.org.
Originally published in *Progressions: The Peer-Led Team Learning Project Newsletter*, Volume 10, Number 2, Winter 2009.