Self Direction & Constructivism in Programming Education

Naomi R. Boyer, Sarah Langevin, Alessio Gaspar

University of South Florida Polytechnic
3433 Winter Lake Road, Lakeland, FL, 33803 USA
[Naomi | Sarah | Alessio] @ softice.lakeland.usf.edu

ABSTRACT

This paper explores the relationship between new constructivist apprenticeship techniques meant to improve programming pedagogy **[6][7]** and student self-direction. To this end, we used the lens of the Personal Responsibility Orientation **[2]** to measure the impact on student self-efficacy and self direction of our interventions. These learning activities were introduced based on peer learning and authentic student feedback principles. They consisted of peer learning weekly forums and student-led "live coding" hands-on exercises. These were applied to both an introductory (cop2510 **[14]**) and intermediate (cop3515 **[5]**) programming courses. Results derived from an online anonymous survey are presented and interpreted.

Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]: Computer Science Education / Information Systems Education

General Terms

Design, Human Factors, Languages

Keywords

CS-1, Introductory Programming Courses, Self Directed Learning

1. INTRODUCTION

Our research aims at studying the relations between self-direction, constructivist apprenticeship, and programming skills. This work is motivated by the fact that computing professionals are required to leverage self direction in their life-long learning in order to adapt to new emerging technologies. Similarly, the creative nature of programming requires students to often think outside of the box and investigate alternative solutions on their own in order to acquire genuine programming skills with higher cognitive capabilities (i.e. with respect to Bloom's Taxonomy). Paradoxically, the role of self direction as a predictor of success in programming courses, or as a way to help student efficiently strategize their learning, has not been explored to date.

SIGITE 2008, October 16-18, 2008, Cincinnati, OH, USA

1.1 Defining Self Direction

For the purposes of this study, the Personal Responsibility Orientation model **[2]** will be used as a foundation for investigating self-directed behaviors in programming courses. Brockett & Hiemstra describe self-direction as a combination of process and personal elements in which an individual "assumes primary responsibility for a learning experience" (pg. 24). Within their model, despite the emphasis placed on the internal characteristics of the individual, the social context also plays a critical role surrounding the learning experience. Self-direction is not a new concept **[15]** and many have attempted to find ways to incorporate strategies to encourage self-directed behaviors within certain learning environments and disciplines. The concept of self-directed learning is aligned with any delivery, content area, and context of learning. In a nationwide study, 64% of businesses indicated that the applied skills of lifelong learning/self-direction were expected to have an increasing importance over the next five year **[4]**. While 78.3% businesses felt that lifelong learning/self-direction were very important in the workforce, only 25.9% rated 4-year college graduates as excellent in this area. These concepts are important in all fields, even more so in IT education.

1.2 Defining Programming

In this work, we will consider teaching programming through its impact on enabling students to solve computing problems by analyzing requirements, designing and implementing an algorithmic solution in a programming language and evaluating its correctness. Each components of this cognitive process falls into one of the following categories.

Factual Knowledge encompasses many different types of programming concepts such as definitions (e.g. what is a program, what is a statement, what is an algorithm), syntactical rules in a given programming language (e.g. C or Java), programming building blocks (e.g. conditional and iterative statements), programming patterns (e.g. which type of loop to use to solve a specific category of problems), and understanding of how programs execute and are interpreted by computers.

Programming skills involve the capability of translating in plain English, and oftentimes ambiguous description of a computing problem, into a solution. This solution is first designed, using abstract notations (e.g. flowcharts, pseudo code), and then implemented in a given programming language (e.g. C or Java). Programming skills also require students to be able to evaluate the correctness of their solution and justify each part of it as being a step toward the achievement of the stated goals.

1.3 Problem Statement

Due to the many components of the programming thought process, even a novice programmer needs to have a foundation in all of the above-mentioned types of knowledge and skills before be able to program solutions to computing problems. This leads to a "bootstrapping" problem when trying to design a programming pedagogy; do we need to teach each skill and knowledge separately and in a specific order, introducing them only as needed? Would it be better to teach a minimal level in each knowledge and skills and then further study them in turns?

When facing this didactic dilemma, the former approach is often favored. Many textbooks have introduced students to programming by focusing on concepts and definitions first, thus encouraging rote learning. While acceptable for the factual aspects of programming, this often also encourage students to practice by only cutting and pasting already written (and correct) programs or by simply "filling in the blanks" in already almost functional programs. The idea behind this instructional technique is inspired by methods used to teach foreign languages; students are first "immersed" in the language through conversations which, later on, motivate the formal acquisition of grammatical structures and vocabulary. While a contextualization of knowledge is undeniably a sound strategy, this particular approach can lead to a "loss of intentionality" in novice programmers **[6]**.

This loss of intentionality can be described by students equating "programming" with "using and adapting others' programs". This encourages them to believe they don't need to understand programs in-depth anymore to be able to re-use them to solve new problems. Later, this misconception leads them to face new problems without any problem solving thought process but with a pattern matching philosophy instead. Every new problem can be related to the description of a pre-solved problem they studied. The solution for the latter is then regurgitated as a first attempt at solving the new problem. When this fails, the program is modified to fit the new requirements. Because of the lack of in-depth understanding of the programming activities or concepts, the modifications end up being almost random (if students had a good grasp on both concepts and skills they'd design their own solution to start off with). This is further exacerbated by the ease with which students can compile and test their programs and have automated tools point out errors to them. This entire though process is based on (1) pattern matching and (2) random modifications based on automatic feedback. This is disturbingly close to evolutionary computation's genetic programming **[11]**.

This issue also arises in courses where instructors make students practice their programming skills by showing them a slide describing a problem to solve, letting them work on it for a few minutes, and then commenting abundantly on the listing of the program implementing a correct solution. While there is clearly an attempt here at teaching students how to solve problems from scratch, this approach also reinforces the loss of intentionality. This is especially true for students who have difficulty grasping the programming thought process. These seize this opportunity to build a "dictionary" of problems-solutions pairs. When facing a new problem, they once again attempt to match its requirements to already solved problems they collected and proceed with adapting it more or less randomly. In either case, the outcome is a pedagogy which does not teach students how to program but rather teaches them about programming.

1.4 Traditional Educational Responses

The lack of focus on the programming though process itself can be addressed by using cognitive apprenticeship **[3]**. This approach can be illustrated by the BlueJ programming environment **[10]** and the didactic developed by its authors in their textbook **[1]**. The key of these approaches is to have the instructor demonstrate, preferably "live", how (s)he would solve a given problem by developing a solution from scratch. Students are generally responsive to this pedagogical strategy which realigns what is taught with the real learning outcomes; programming skills themselves. Students are exposed repeatedly to the thought process of the instructor which they can attempt to mimic or internalize instead of simply being shown the final result (complete working program) of the programming activity.

This approach is further improved by a recent emerging trend which attempts to incorporate test-driven professional development practices to novice programmers **[13]**. Test-driven software development methodologies are based on the idea that a test-harness should be developed prior to the development of the code that solves a given problem. A test-harness can be seen as a list of test cases which consist of a set of input values for the program and a list of expected outcomes. Test-harness can be implemented as programs or used as a list for educational purposes. A program's correctness is later on assessed by running it for each test case and making sure the correct outcome is produced. This approach is interesting from a pedagogical perspective in so far that it can complement the traditional learning activities focused on having students develop solutions by activities which help them overcome the well documented issues they have with deciding when their solutions are correct **[9]**. While these efforts have been successful in overcoming the learning barriers discussed in the previous sections, their limitations provide room for improvement.

1.5 Pedagogical Specificities of our offerings

Our work focused on enhancing the pedagogy of instruction in introductory and intermediate programming courses at both the factual knowledge understanding and the programming skills acquisition levels. This was achieved by developing learning activities embedding two educational strategies.

First, we developed peer learning activities with the expectation that it would provide a learning environment under which students would complement the instructor-led teaching by challenging each other. Our main motivation came from the assumptions that (1) there are differences in expertise among the students in the class and that (2) the difference between two arbitrary students is on average smaller than between these students and the instructor. Under these assumptions, we conjectured that peer learning programming activities would challenge students in their zone of proximal development (ZPD) **[17]**. This can facilitate the design of suitable exercises since it can be daunting otherwise for an instructor to adapt the pedagogy of instruction to each and every student's ZPD.

Second, we re-organized instruction based on an authentic feedback model. Instead of basing pedagogy of instruction on the instructor's previous personal experience, published literature (textbook) or published discipline-based education research, we base it on the authentic learning barriers encountered by the particular student population being taught. This is enabled by the

student-centered nature of the learning activities we implemented in our courses. As we will describe below, students' errors and issues with specific learning barriers are explicated by these activities and thus allow the instructor to react to them and then adapt accordingly the method and content of instruction. It is worth mentioning that this meta-strategy can prove superior to simply adopting any didactic which proved efficient on a given student population. It is common to find successful strategies in the computing education literature, which quantitative impact on student learning has been measured with statistical significance. However, it is rare to see studies which provide enough information on the student population to infer whether the results are applicable at other institutions. For instance, a successful strategy applied to Stanford's full time students might not be applicable to evening courses at a small campus, regardless of the mathematical significance of the published statistics. Building in authentic feedback from students allows to flexibly adapting teaching methods to the real learning barriers encountered by the students being taught without the need for (educated) guesses.

Following these two strategies, we implemented two learning activities. First, Blackboard learning management system's discussion room features were used to engage students in weekly discussion-based learning activities. After each class meeting, students were given two days to read an assigned chapter (or a video to watch). During this period, they would be responsible for posting questions on the forums about anything unclear. This participation was assessed and graded. After this first period, students were invited to read all questions and pick a couple they would attempt to respond to based on their own understanding of the assigned material. To conclude this activity, students ranked questions according to how much they wanted them to be discussed during class time. At the next class meeting, the instructor designed a lecture based on the issues posted on the forums and answers. Besides encouraging peer learning dynamics of the factual knowledge, this activity also responded to the need for students to learn to read and understand technical information on their own. While required from computing professional and graduate students, this skill is seldom taught at undergraduate level where it's all too common for the "sage on the stage" to almost read aloud the text to students. This is a barrier to develop self-efficacy of students which end up assuming they need to be guided to acquire more information.

The second instruction intervention utilized peer learning activities meant to develop a student's programming skills. Cognitive apprenticeship methods were modified and new constructivist elements were introduced. When an instructor shows students how to develop solutions from scratch, the focus of the teaching effort is aligned with the learning outcome. However, the manner in which students are taught the programming thought process is instructivist in essence; students are passively watching the instructor demonstration just as they watch lectures in other courses. We developed, as part of a constructivist apprenticeship strategy **[6]**, programming activities which are student-led. The philosophy of constructivism assumes that "human learning is *constructed*, that learners build new knowledge upon the foundation of previous learning. This view of learning sharply contrasts with one in which learning is the passive transmission of information from one individual to another, a view in which reception, not construction, is key" **[8]**. A student is picked and given a wireless keyboard and mouse set connected to the podium PC which screen is projected for all to see. For the duration of the exercise, this student will work out his or her solution in front of the other students. This activity develops critical thinking, troubleshooting and other programming skills related to the evaluation of the correctness of the solution being developed. Unlike instructor-led approaches, this activity exposes the students' thought process thus enabling an apprenticeship learning which is guided by the authentic learning barriers encountered by these specific students.

2. Method

2.1 Sample

This study was conducted with 15 students enrolled in junior level programming courses during the fall, 2007. Preliminary work was done during the spring, 2007, which focused on the teaching methods employed in this study. There were eight in the introductory course and another seven students in the intermediate computer programming course. Most students in these courses have transitioned from the community college into the university following the 2+2 model established within the State of Florida. Students in general are non-traditional in nature, taking evening courses and working during the day. Tremendous variation exists in the age of this student population, as some of these students may have graduated from high school with the community college degree and others may be more mature adults returning for further education beyond technician type credentials.

The introductory course, COP 2510 Programming Concepts, is a first-time programming course for Information Technology, Computer Science, Computer Engineering, and a collection of other majors and is labeled as cop2510 when referred to throughout the paper. It uses the Java programming language with a "fundamentals first" approach **[14]**. The intermediate course, COP 3515 Program Design, is meant as a follow-up on the latter and is taught to IT majors only. The C language was used for this course to strengthen students' skills and expose them to low-level concepts (program stack, heap…) to prepare them for system-oriented senior-level courses (e.g. operating systems). A classical text from Deitel Associates is used for this course **[5]**. For some students, cop3515 is their second exposure to the teaching techniques utilized in the course.

2.2 Instrumentation

The PRO-SDLS "Learning Experience Scale" **[16]** instrument was utilized as a basis for the survey that was designed to attempt to capture the level of self-direction reported by students after participating in the computer programming course experience. The scale consists of 25 questions representing two subcomponents: teaching learning transaction component and learner characteristic component. Within these two subcomponents are four factors: initiative, control, self-efficacy, and motivation. Likert scale responses were used for these questions and represented the values strongly disagree (1) to strongly agree (5). Total possible score on the instrument is 125. The initiative, control, and self-efficacy factors have a maximum sum score of 30 with the motivation factor having a maximum sum score of 35. Questions were slightly altered to respond to the particular educational context and a post intervention administration. The online instrument was administered during class activities and was embedded within other course specific questions that inquired as to the overall learning experience.

2.3 Procedures

To gather information about the successfulness of this technique for facilitating student learning and the level of increased self-directedness, the PRO-SDLS (2006) was administered as part of a larger survey set via SurveyMonkey to determine the student's level of personal responsibility for the learning process. Students were provided with the web link and asked to anonymously complete the instrument during the final class session. While participation in the study was voluntary, students were strongly encouraged to complete the survey/instrument. A general open ended question was included in the cop2510 survey. The completion of the overall survey took students approximately 15 minutes, with a range from 8-20 minutes across all students.

2.4 Data Analysis

The data collection was facilitated electronically using SurveyMonkey. The online tool provides basic frequency information. The data were then transferred into other software packages for further descriptive analysis. General means were run for each question for each section and then the data combined for a global perspective on the issue. Given that the open ended question was only included for the beginning programming course, the number of responses is significantly lower, thereby only providing only minimal fodder for analysis.

3. Major findings

A total of fifteen students responded to the survey and completed the PRO-SDLS. Mean raw scores (with std. dev. in parentheses) for the beginning (cop 2510) and intermediate (cop3515) courses were 93.75 (13.38) and 85.00 (8.93) respectively. Combined, the resulting mean was 89.67 (12.00). Overall, cop2510 had distinctively higher scores with much greater variation. A complete listing of the descriptive statistics of raw scores can be found in Table 1 including minimum and maximum values.

	Mean	Std dev	Min	Max
cop 2510 (N=8)	93.75	13.38	72.00	112.00
cop 3515 (N=7)	85.00	8.93	71.00	95.00
Both	89.67	12.00	71.00	112.00

Table 1: Descriptive Statistics for PRO-SDLS scores

The questions from the instrument can be found in **[16]**. The mean (4.13) on the five point scale for question 12 suggests that the majority of students are convinced that they have the ability to take control of their own learning. In addition, most students indicated that they would spend additional time learning about this topic after completion of the course (mean-4.13). The cop2510 students (means noted in parentheses) indicated a level of relevance in the course work (4.29), an ability to independently find (4.29) (reverse scored) and use (4.14) materials outside of the class applicable to the topic, and an ability to carry out their student plan (reverse scored) (4.14). Further, this group also felt confident in their ability to consistently motivate themselves (4.14) and independently prioritize their goals (4.29), do extra work because of a personal interest (4.14), connect course work and personal goals (4.29) (reverse scored), work independently to make changes to improve in the class (4.00), take responsibility for their own learning (4.43), learn new things on their own rather than wait for the instructor (4.14), and take personal control over their learning (4.57). The cop3515 students knew why they completed the work that they did (4.14) (reverse scored).

Students in cop2510 actually had higher SDLS scores than those in cop3515 for almost all questions. The only factor that had lower values for cop2510 was motivation at a mean of 20.63 for all motivation questions vs. the cop3515's value of 2.14. Table 2 provides information on the factors by component and course.

Teaching Learning Transaction Component								
Initiative	2	9	10	15	17	25		**Total**
cop 2510	4.00	4.13	4.13	4.50	3.88	3.50		24.13
cop 3515	3.14	2.86	3.29	3.71	3.43	2.86		19.29
Control	4	5	6	13	19	23		**Total**
cop 2510	4.00	4.25	3.50	2.75	4.00	3.50		22.00
cop 3515	3.00	3.57	3.57	3.14	3.43	3.29		20.00
Learner Characteristics Component								
Self-Efficacy	1	7	12	21	22	24		**Total**
cop 2510	4.00	4.25	4.38	3.63	4.13	3.88		24.25
cop 3515	3.14	3.43	3.86	3.86	3.43	3.43		21.14
Motivation	3	8	11	14	16	18	20	**Total**
cop 2510	4.00	3.88	3.75	4.25	2.13	3.38	3.25	20.63
cop 3515	3.86	3.57	4.14	3.57	2.71	3.29	3.86	21.14

Table 2: Question Means by course for Each PRO-SDLS Component and Factor (Note; same N values than other tables).

An open ended question was only provided to the students in cop2510. Students shared comments that indicated an appreciation for the instructional methods and an ability to take responsibility for their own learning process and outcomes. The open ended question was as follows: "Provide any complementary feedback on how this course's pedagogies have influenced your self-direction in learning". Of the eight students in cop2510, six responded and their comments can be found in Table 3 below.

Student Comments
I love to program more, and I plan to do more programming in java.
G[ave] me the ability to motivate myself to a better perspective learning cycle.
The course influenced me because it is clear that in order to solve a problem the steps needed to be followed.
The teaching style in this course has been extremely helpful to me personally beyond just the course.
I have become more motivated to do my homework.
I really had to push myself to read the material and participate in the forums on the deadlines.

Table 3: Student Comments to Open-Ended Prompt on the Topic of Self-Direction (Note: grammatical and spelling alterations).

4. Discussions

The potential of using the above-mentioned peer learning activities in introductory and intermediate programming courses in order to help students overcome learning barriers, such as a loss

of intentionality when designing computer programs, is very important for computing education. Organizational out-sourcing and industry demands have stimulated discussion in higher education about how to attract, retain, and graduate successful information technology professionals that can respond to complex computing needs and continue to learn from the constantly changing and evolving demands of technology. To this end, new instructional andragogical techniques have emerged from the field that influence a student's ability to self-direct when faced with new technical issues that require a programming response. Andragogy refers to the methods of teaching adults who are different in development capacity, rather than the traditional pedagogy that is implemented with adults in the higher education setting **[12]**. There are a number of limitations that should be considered when reviewing the results of this exploratory study. The small number of participants in this study, limits any sort of generalization and or strong conclusions. This is the first semester of data collection, which will be expanded upon in future semesters. In addition, the PRO-SDLS was administered at the end of the semester to capture the student ratings of their increase or decrease in self-directed behavior as a result of the class. To capitalize on the type of information gathered as part of the PRO-SLDS a pre and post administration would allow students to assess their level of self-direction at the initiation and then the conclusion of the course to look for changes in behavior and perception. The instrument was piloted during this phase of the study and will require some additional adjustment for clarity and validity of concepts. The student open-ended comments were solicited in only one course. Due to the relevance of this information it will be included in each future administration.

Despite these limitations, a few initial observations can be offered in regard to how the teaching method impacted the students' learning and self-direction. In general the student comments (offered only in one course) indicated an appreciation for the long term benefit of the instructional methods. Students expressed the use of self-directed strategies in other courses.

The scores (means) on the PRO-SDLS are within the moderate to high range for each of the four factors with self-efficacy receiving the highest overall scores for both courses. Table 4 indicates how each of the factors were characterized as high, medium, or low and the associated mean scores for each course based upon this assumption. The cop2510 course had high scores for initiative and self-efficacy, and moderate scores for control. The only score that was lower for cop2510 than cop3515 was motivation. There were no scores that stand out as high for cop3515 and there can be no judgments made about each of the following two components; teaching learning transaction and the learning characteristics.

Several factors need to be taken into consideration when interpreting the scores results in both courses. Firstly, cop2510 is a first programming course for most of the enrolled students. As such, its student population comprises individuals who will realize they are not interested in (or don't have an intellectual affinity with) programming. This course also generally includes students from engineering, information technology, computer science and computer engineering, thus providing for a variety of perspective on the usefulness of programming for the student's future career. In such a context, the fact that students indicated that the course's pedagogies resulted in a high self-efficacy with respect to programming and a high initiative, is extremely rewarding and indicate that the teaching methods employed might also have benefits in terms of motivating first-time programmers to overcome their learning barriers with a new discipline. Given the nation-wide enrollment decreases in computing disciplines, we think that these methods should be further studied from this perspective. It would be particularly interesting to compare our approaches to other pedagogies currently used to attract students in introductory computing courses but which often rely heavily on multi-media and three dimensional interactive environments. While extremely motivating, such methods often depict a picture of the computing discipline which higher-level courses, bound to present more technical and difficult aspects, might not be able to sustain. This might results in attraction vs. long term retention.

Secondly, it might be expected that students enrolled in cop3515 will have already had opportunities to mature their learning strategies. As such, their perception of the benefits of the instructional methods used might not be as enthusiastic in so far that they might consider these learning routines as "common sense" rather than feel they are something new and beneficial.

Thirdly, cop3515 comprised only information technology majors while, as discussed above, cop2510 was more heterogeneous. It is not impossible that the observed differences might be indicators of significant differences in the student populations which have not been captured by our current instrument. The next measures will include basic demographic information as well as data regarding the student's majors as well as their curricular and extra-curricular workloads during the semester. Many IT students have significantly different age and occupational profiles which might explain differences in their learner profiles. Full time workers, in particular, might be under constraints which prevent them from participating to the learning activities which therefore might not appear as useful as they could to them.

	Teaching Learning Transaction Component		Learner Characteristics Component	
Category	Initiative	Control	Self-efficacy	Motivation
# questions	6	6	6	7
Value Ranges	High 24-30 Moderate 15-23 Low-6-14	High 24-30 Moderate 15-23 Low-6-14	High 24-30 Moderate 15-23 Low-6-14	High 28-35 Moderate 16-27 Low-7-15
cop 2510 (N=8)	24.13 High	22.00 Moderate	24.25 High	20.63 Moderate
cop 3515 (N=7)	19.29 Moderate	20.00 Moderate	21.14 Moderate	21.14 Moderate

Table 4: Learning component, factor characterization, associated scale

It is also worth taking into consideration that, besides their impact on the teaching and learning of programming, the instructional activities described in this paper also worked on developing learning skills which are indispensable to computing professional. Among these, the most important is the ability to self-direct one's learning to adapt to an ever changing technological landscape. The peer learning forums activities helped scaffolding the development of technical reading skills. These will be relevant during our students' computing career regardless of whether they have to program or not. While critical to professional and graduate students alike, this skill is seldom practiced, let alone with supports for progression through their personal ZPD, in undergraduate courses. This makes the peer learning forum activities worth further investigating on their own.

There are a number of questions resulting from this initial phase of study that will be explored in future phases of this research. While it is interesting that cop2510 students had overall higher scores than cop3515 ones, it is unclear if this is only specific to this small group or if a trend will develop. One contributing element may be that cop2510 includes both engineering and information technology students, while cop3515 is usually made up of only information technology students. Do IT students rate lower in personal responsibility for self-direction? As was previously mentioned, continuing this research to increase the number of participants will further enhance the study; however, there are also plans to expand these approaches to students in other disciplines. As a more general future study, it would be beneficial to the IT field to explore how self-direction in IT workers impacts employee performance in the field.

In addition, further work in assessing the impact of the use of these techniques on the overall learning of programming are important to determining the success or detraction from achieving course objectives. Essentially, do the students also learn more as a result of the intervening instructional method? Learning in this type of study would need to be carefully defined in order to determine if depth, breadth, or critical thinking is the original intention of the programming experience.

The use of constructivist apprenticeship, live coding, and antagonistic programming activities is extremely flexible and can benefit beyond programming offerings on which we have been focusing our discussion so far. Any course conveying a problem-solving skill to students can benefit from these andragogical strategies (e.g. accounting, software engineering, algorithms design…). In this expanded context, "live coding" and "code peer review" activities can be more broadly perceived as "peer reviewed problem solving", which then leads to the capacity for self-direction within the broader context of problem solving.

5. REFERENCES

[1] Barnes, J., Kolling, M. (2006). Objects First With Java: A Practical Introduction Using BlueJ (3rd Edition), Prentice Hall, River Saddle NJ

[2] Brockett, R. G. & Hiemstra, R. (1991). Self-direction in adult learning: Perspectives on theory, research, and practice. London and New York: Routledge.

[3] Collins, A., Brown, J. S., & Newman, S. E. (1987). Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics (Technical Report No. 403). BBN Laboratories, Cambridge, MA. Centre for the Study of Reading, University of Illinois. January, 1987.

[4] Conference Board, Corporate Voices for Working Families, Partnership for 21st Century Working Skills, & Society for Human Resource Management. (2006). Are they really ready to work? Employer's perspectives on the basic knowledge and applied skills of new entrants to the 21st century U.S. workforce. Retrieved December 17, 2007, from http://21stcenturyskills.org/documents/FINAL_REPORT_PDF09-29-06.pdf

[5] Deitel, H & Deitel, P. (2006). C How to program, 5/e, Pearson Education, Prentice Hall, Upper Saddle River NJ 07458, ISBN-10: 0-13-240416-8

[6] Gaspar, A., Langevin, S., Boyer, N. (2007). Constructivist Apprenticeship through Antagonistic Programming Activities, Encyclopedia of Information Science & Technology, 2/e, under review.

[7] Gaspar, A., Langevin, S. (2007b). Restoring "Coding With Intention" in Introductory Programming Courses, SIGITE 2007, proceedings of the international conference of the ACM Special Interest Group in Information Technology Education, July 12-15, Orlando, FL (IN PRINT)

[8] Hoover, W.A. (1996). The practice implications of constructivism, SEDL Letter, Vol. IX, No. 3.

[9] Kolikant, Y.B.D. (2005). Students' alternative standards for correctness, Proceedings of the international workshop on computing education research ICER

[10] Kolling, M., Quig, B., Patterson, A., Rosenberg, J. (2003). The BlueJ system and its pedagogy, Journal of Computer Science Education, special issue on learning and teaching object technology, Vol. 13, No. 4, 12/2003

[11] Koza, J.R. (1992). Genetic programming: on the programming of computers by means of natural selection, MIT Press

[12] Knowles, Malcom (1990). A theory of adult learning: adragogy'. In Knowles, Malcom ed., *The Adult Learner: a Neglected Species*, 4th ed., pp. 27-65. Gulf Publishing Company, Houston,TX.

[13] Langr, J. (2005), Agile Java: Crafting code with Test Driven Development, Pearson

[14] Liang, Y.D. (2006). Fundamentals First Introduction to Java Programming, 6/e, Prentice Hall, Upper Saddle River, NJ 07458, ISBN: 0-13-223738-5

[15] Merriam, S.B., & Brockett, R. G. (1997). The professional and practice of adult education. San Francisco: Jossey-Bass.

[16] Stockdale, S. L., & Brockett, R. G. (2006). The continuing development of the PRO-SDLS: An instrument to measure self-direction in learning based on the personal responsibility orientation model. Paper presented at the 20th International Self-Directed Learning Symposium, Cocoa Beach, FL.

[17] Vygotsky, L.S. (1978). Mind and society: The development of higher mental processes. Cambridge, MA: Harvard University Press.