Computer Programming meets Education: The Intersection of Information Technology and Self-Direction

Paper Presentation Proposal

Naomi Boyer, Sarah Langevin, & Alessio Gaspar
3433 Winter Lake Rd
Lakeland, FL 33803
863 667-7088

I warrant that my paper is an original work that has not been previously presented or published. Furthermore, if my proposal is accepted I will attend the symposium and provide an advanced copy of my paper no later than January 5, 2008. I will complete advanced registration no later than January 2, 2008.

Introduction

At long term, our research aims at studying the causal links between programming skills, constructivist apprenticeship and self direction. This work is motivated by the fact that computing professionals are required to leverage self direction in their life long learning in order to constantly adapt to new emerging technologies. Similarly, the creative nature of programming requires students to often think outside of the box and investigate alternative solutions paths on their own in order to acquire genuine programming skills with higher cognitive capabilities (i.e. higher in 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.

The purpose of this presentation is to explore the relationship between new constructivist apprenticeship computing education techniques used to teach effectively programming and student self-direction. To this end, we used the lens of the Personal Responsibility Orientation (Brockett & Hiemstra, 1991) to measure the impact on student self-efficacy and self direction within the context of the newly introduced computing education methods. These constructivist apprenticeship 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 and intermediate programming course. Results derived from an online anonymous survey are presented and interpreted.

Background

Defining Self-Direction

For the purposes of this study, the Personal Responsibility Orientation model (Brockett & Hiemstra, 1991) will be used as a foundation for investigating self-directed behaviors with the information technology discipline. Brockett & Hiemstra describe self-direction as a combination of process and personal elements that 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. The concept of self-direction is not a new one (Merriam & Brockett, 1997) 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 (Conference Board, Corporate Voices for Working Families, Partnership for 21st Century Working Skills, & Society for Human Resource Management, 2006). 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, but are particularly critical for information technology specialists, in particular computer programmers.

Programming: What is it all about?

Teaching programming is about enabling students to solve computing problems by analyzing requirements, designing and implementing an algorithmic solution in a programming language and evaluating its correctness. From the educational perspective, each component of this cognitive process falls into one of the following categories: factual knowledge about programming or programming skills. 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.

The process of programming itself is essentially iterative in nature. Students sketch out a design, implement a small portion of it and immediately check it for syntactical and logical correctness by testing the program against the stated objectives. Once satisfied that they have effectively broken down the overall problem into steps and then successfully implemented a program, they re-iterate this process to tackle yet another step of the problem.

Traditional Programming Educational Challenges

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

When facing such a 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 this might be acceptable for the factual aspects of programming, this approach has also led to teaching the programming process itself by having students only cut and paste already written (and correct) programs to practice with or have them simply "fill in the blanks" in already functional pre-made 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 had an unexpected effect on the development of programming skills as has emerged in previous studies (Gaspar & Langevin, 2007), which underlined how this approach can actually lead to a phenomenon called “loss of intentionality” in novice programmers.

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, it is disturbingly close to the way recent evolutionary computation techniques allow programs to design other programs using a technique known as “Genetic Programming” (Koza, 1992).

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.

Transitioning From Traditional Programming Educational Methods

The lack of focus on the programming though process itself can be addressed by using cognitive apprenticeship (Collins, Brown & Newman, 1987) approaches. This approach can be illustrated by the BlueJ programming environment (Kolling, Quig, Patterson & Rosenberg 2003) and the didactic developed by its authors in their textbook (Barnes, 2006). 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 (Langr, 2005). 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 (Kolikant, 2005). While these efforts have been successful in overcoming the learning barriers discussed in the previous sections, their limitations provide room for improvement.

Innovative methods for teaching computer programming

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) (Vygotsky, 1978). 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 more common for the sage on the stage to "read aloud" the text to students. This in itself can be seen as a barrier to develop the self-efficacy of these students in a topic for which they 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 (Gaspar, Langevin & Boyer, 2007), programming activities which are student-led. 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 this specific student group.

Method

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 the beginning section when referred to throughout the paper . It uses the Java programming language with a “fundamentals first” approach (Liang, 2006). The intermediate course or intermediate section, 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 (Deitel, 2006). For some students in the intermediate section, this would have been their second exposure to the teaching techniques utilized in the course.

Instrumentation

The PRO-SDLS “Learning Experience Scale” (Stockdale & Brockett, 2006) instrument was utilized as a basis for the survey that was designed using SurveyMonkey 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 align to a post intervention administration. The online instrument was administered during class activities and was embedded within other course specific evaluation questions that inquired as to the overall learning experience.

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 beginning section course survey. The completion of the overall survey, inclusive of the instrument, took students approximately 15 minutes, with a range from 8-20 minutes across all students.

Data Analysis

The data collection was facilitated electronically using a University account within 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.

Major Findings

A total of fifteen students responded to the survey and completed the PRO-SDLS. Means (with standard deviations in parentheses) for the beginning and Intermediate sections were 93.75 (13.38) and 85.00 (8.93) respectively. Combined the resulting mean was 89.67 (12.00). Overall the beginning section had distinctively higher scores with much greater variation. A complete listing of the descriptive statistics can be found in Table 1 including minimum and maximum values.

Table 1

Descriptive Statistics for PRO-SDLS scores

	Mean	St. Deviation	Minimum	Maximum
Beginning Section (N=8)	93.75	13.38	72.00	112.00
Intermediate Section (N=7)	85.00	8.93	71.00	95.00
Combined Sections	89.67	12.00	71.00	112.00

The questions from the instrument and the descriptive statistics by question can be found in Appendix A. The mean (4.13) on 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 beginning section (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 intermediate group knew why they completed the work that they did (4.14) (reverse scored).

Students in the beginning section of the course actually had higher SDLS scores than those in the intermediate section, for all questions except for a handful of question. The only factor that indicated lower values for the beginning section was motivation at a mean of 20.63 versus the intermediate section value of 2.14. Table 2 provides information on the factors by component and section.

An open ended question was only provided to the beginning section of the class. 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:

Table 2

Question Means by Section for Each PRO-SDLS Component and Factor

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

Note. The intermediate section N=7 for all factors and the beginning section N=8.

"Provide any complementary feedback on how this course's pedagogies have influenced your self direction in learning". Of the eight who were in this section six responded and their comments can be found in Table 3 below.

Table 3

Student Comments to Open-Ended Prompt on the Topic of Self-Direction

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

Note: Some grammatical and spelling corrections have been made to the comments above.

Conclusions

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.

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 section of the class and this information is extremely valuable and will need to 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 learning and self-direction of programming for the student. In general the student comments (offered only in one class section) 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 the self-efficacy factor receiving the highest overall scores for both sections. Table 4 indicates how each of the factors were characterized as high, medium, or low and the associated mean scores for each section based upon this assumption. The beginning section had high scores in the areas of initiative and self-efficacy, and moderate scores for control. The only score that was lower for the beginning section than the intermediate section was motivation. There were no scores that stand out as high for the intermediate section 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 sections. Firstly, the beginning section 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 form 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 environment. 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 thus favoring the attraction of students to the discipline vs. their long term retention.

Secondly, it might be expected that students enrolled in the intermediate section will have already had opportunities to mature their learning strategies. As such, their perception of the benefits of the instructional methods we used might not be as enthusiastic due to the fact they might consider these learning routines as "common sense" rather than feel they are something new that might benefit them.

Thirdly, the intermediate section comprised only information technology majors while, as discussed above, the beginning section 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 they are taking our courses. 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.

Table 4

Learning component and factor characterization and associated scale

	Teaching Learning Transaction Component		**Learner Characteristics Component**	
	Initiative- 6 Questions	**Control- 6 Questions**	**Self-efficacy- 6 Questions**	**Motivation- 7 Questions**
	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
Beginning Section (N=8)	24.13- High	22.00-Moderate	24.25-High	20.63-Moderate
Intermediate Section (N=7)	19.29- Moderate	20.00- Moderate	21.14- Moderate	21.14-Moderate

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 capability to self direct one’s learning in order to be able to adapt to an ever changing technological landscape. The peer learning forums activities, more specifically, helped scaffolding the students’ development of technical reading skills which are going to be critical during their computing career regardless of whether they have to program or not. While critical to professional and graduate students alike, this skill is seldom practiced and scaffolded in undergraduate courses thus making 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 the beginning section had overall higher scores than the intermediate section, it is unclear if this is only specific to this small group or if a trend will develop. One contributing element may be that the beginning section includes both engineering and information technology students, while the intermediate course is usually made up of only information technology students. Do information technology 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 field of information technology to explore how self-direction in information technology workers impacts employee performance in the field.

The use of constructivist apprenticeship, live coding, and antagonistic programming activities is extremely flexible and can benefit beyond the programming courses (introductory and intermediate) we have been focusing our discussion on so far. Any course conveying a problem-solving skill to students can benefit from these andragocial 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.

References

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

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

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.

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

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

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

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)

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

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

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

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

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

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

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.

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

Appendix A

General Prompt: As a result of taking this course… Questions	Beginning Section Mean (SD)	Intermediate Section Mean (SD)	Combined Section Mean (SD)
1. I am more confident in my ability to consistently motivate myself.	4.14 (.90)	3.14 (.69)	3.60 (.91)
2. I frequently do extra work in this course just because I am interested.	4.14 (1.21)	3.14 (.90)	3.60 (1.12)
3. I still don't see any connection between the work I do in this course and my personal goals and interests.	4.29 (1.25)	3.43 (.79)	3.73 (1.16)
4. If I am not doing as well as I would like in this course, I more often independently make the changes necessary for improvement.	4.00 (.82)	3.00 (1.41)	3.53 (1.19)
5. I more often and more effectively take responsibility for my own learning.	4.43 (.53)	3.57 (.79)	3.93 (.80)
6. I more often have a problem motivating myself to learn	3.71 (1.38)	3.57 (1.40)	3.53 (1.36)
7. I became very confident in my ability to independently prioritize my goals.	4.29 (.76)	3.43 (.98)	3.87 (.92)
8. I complete most of my college activities because I WANT to, not because I HAVE to.	3.86 (1.35)	3.57 (.79)	3.73 (1.03)
9. I would rather take the initiative to learn new things In a course rather than wait for the instructor to foster new learning.	4.14 (1.07)	2.86 (.90)	3.53 (1.13)
10. I more often use materials I have found on my own to help me in a course.	4.14 (.90)	3.29 (.95)	3.73 (.96)
11. For most of my classes, I still really don't know why I complete the work I do.	3.86 (1.07)	4.14 (.69)	3.93 (.88)
12. I am very convinced I have the ability to take personal control of my learning.	4.57 (.53)	3.86 (1.07)	4.13 (.92)

Continued

Question	Beginning Section Mean (SD)	Intermediate Section Mean (SD)	Combined Section Mean (SD)
13. I still usually struggle in classes if the professor allows me to set my own timetable for work completion.	2.86 (1.07)	3.14 (1.21)	2.93 (1.10)
14. I feel that most of the work I do in my courses is personally enjoyable or seems relevant to my reasons for attending college.	4.29 (.76)	3.57 (.98)	3.93 (.88)
15. Even after this course is over, I will continue to spend time learning about the topic.	4.57 (.79)	3.71 (.76)	4.13 (.83)
16. The primary reason I still complete requirements is to obtain the grade that is expected of me.	1.57 (.79)	2.71 (1.11)	2.20 (1.08)
17. I more often collect additional information about interesting topics even after the course has ended.	3.29 (1.38)	3.43 (.79)	3.40 (1.06)
18. The main reason I do the course activities is still to avoid feeling guilty or getting a bad grade.	3.00 (1.63)	3.29 (.95)	3.13 (1.25)
19. I am now very successful at prioritizing my learning goals.	4.00 (.82)	3.43 (.53)	3.73 (.70)
20. I still feel most of the activities I complete for my college classes are NOT really personally useful or interesting.	3.57 (1.27)	3.86 (.69)	3.53 (1.19)
21. I am still really uncertain about my capacity to take primary responsibility for my learning.	3.71 (1.38)	3.86 (.38)	3.73 (.96)
22. I am still unsure about my ability to independently find needed outside materials for my course.	4.29 (.95)	3.43 (1.13)	3.80 (1.08)
23. I more often effectively organize my study time.	3.43 (.98)	3.29 (.76)	3.40 (.83)
24. I still don’t have much confidence in my ability to independently carry out my student plans.	4.14 (.90)	3.43 (1.13)	3.67 (1.11)
25. I more often rely on the instructor to tell me what I need to do in the course to succeed.	3.71 (1.11)	2.86 (1.21)	3.20 (1.21)