ISSN 2039-2117 (online)
ISSN 2039-9340 (print)
Mediterranean Journal of Social Sciences
MCSER Publishing, Rome-Italy
Vol 5 No 14
July 2014

Novice Students and Computer Programming: Toward Constructivist Pedagogy

Jacqui Chetty

Glenda Barlow-Jones

Department of Applied Information Systems
University of Johannesburg, Johannesburg, South Africa
Email: jacquic@uj.ac.za and glendab@uj.ac.za

Doi:10.5901/mjss.2014.v5n14p240

Abstract

In order to develop computer programmings skills: critical thought, problem solving, attention to detail, accuracy and abstract thinking are required. Unfortunately, many novice students at universities within South Africa have not developed such skills in their formative years. This is often due to the fact that many of them have been part of a schooling system that does not teach students how to think critically or how to solve problems. Consequently, they find it difficult to acquire these skills at a post-secondary level and they are often at risk of failing computer programming modules. Furthermore, traditional pedagogies used at post-secondary level often do not provide an opportunity for students to develop the skills needed to write programs. Such pedagogies are often teacher-centric, which do not encourage students to develop critical thinking. This paper aims to demonstrate that adopting an alternative pedagogy, namely social constructivism, can assist students in cultivating the skills needed for computer programming. The pedagogy was applied only to students who were at risk of failing a computer programming module. It encouraged a student-centred environment, where active learning, student collaboration and metacognition were promoted. The data clearly indicates that adopting such pedagogy provides an opportunity for students to improve their computer programing skills.

Keywords: *social constructivism; collaboration; active learning; pedagogy; at-risk students*

1. Introduction

Many students in South Africa enter tertiary education under prepared (Jansen, 2012). One of the reasons why this failure rate occurs may be attributed to students matriculating from the government schooling system that has not prepared them adequately for university (Hungi, 2010). These schools often have the following commonalities: they have a very high pupil to teacher ratio; teachers are poorly qualified; schools are often poorly funded; and English, as a medium of instruction, is not pupils' first language and the result is that pupils' written work is below par (Fedderke, 2000). The result of this is that schools often fail to generate adequate pass rates and a large percentage of the students who pass their final school year struggle to pass their first years of university (Jansen, 2012). For example, of students admitted to higher education in 2000, only one in five students graduated in the time envisaged they should take to complete their qualifications (Boughey, 2009). Attrition rates for first year students were 29% and 56% of registered students left higher education without completing their qualification. Most of these students were black students, the very students whom higher education was meant to serve (Boughey, 2009). It is therefore important for lecturers at universities to ensure that students acquire the necessary computer programming skills, as without these skills students cannot compete in such an environment.

Teaching-and-learning of computer programming at universities within South Africa often make use of traditional teacher-centric pedagogies. These pedagogies focus on teacher-centric learning, which comprises of activities, such as lecturing, questioning and demonstration. The lecturer is the expert who transfers knowledge across to students (Xiaohui, 2006). The advantages of such teaching are that it is efficient for large groups and it provides an adequate foundation on which information can be built. However, the disadvantages are that students are passive participants and they have very little chance to interact with peers, tutors or lecturers. They tend to memorise knowledge instead of assimilating and making sense out of the knowledge. Consequently, the emphasis is on the knowledge itself rather than developing students learning skills. However, there are pedagogical approaches to teaching-and-learning that can be easily understood by others. However, there are other pedagogical approaches to teaching programming (Wulf, 2005) that can be used to support students. Such pedagogies can include aspects of social constructivism (Stetsenko, 2002), apprenticeship (Boyer, 2008), collaborative learning (Preston, 2006b) and an off-shoot of collaborative learning known as pair programming (Preston, 2006a). Although these pedagogies are not solely developed with computer programming in mind, research indicates that such pedagogies can improve the teaching-and-learning experience of computer programming, for lecturers and students alike (Preston, 2006b).

Consequently, this article describes how social constructivist pedagogy can be applied so that students may find it easier to understand concepts associated with the development of computer programs. The pedagogy was only applied to students who were found to be at risk of failing a computer programming course. This article firstly describes the difficulties that many students are faced with when entering university. Due to such difficulties the article then secondly describes a support structure that the university has put in place for these students. However, this structure does not include any didactic processes or approaches. Thirdly, the manner in which teaching-and-learning takes place within a department of the university is then described. Fourthly, the proposed approach to teaching at-risk students how to solve problems and develop computer programs is discussed and finally conclusions and recommendations are made.

2. Literature Review

This section looks at the difficulties encountered by computer programming sudents; the teaching-and-learning of computer programming; and an alternative pedagogical approach to teaching-and-learning computer programming.

2.1 Difficulties Encountered by Computer Programming Students

Most of the students within the University of Johannesburg have matriculated from the government schooling system (Van Zyl, 2012), where many of them were subjected to a defunct schooling system (Hungi, 2010). The outcome is that schools often fail to generate adequate pass rates. These commonalities result in pass rates being inflated and pupils not being adequately prepared for university (Jansen, 2012). Furthermore, many students at the university where the study was conducted, originate from a poor socio economic background (Van Zyl, 2012). For example, data collected from students studying at the university in 2012 indicates that 66.2% of students are first generation university participants, where 58.4% of such students have parents who obtained only a matric or less. Fifty three point seven percent of students indicated that English is not their first language. These students are also plagued with many worries, such as poor nutrition, and financial problems. Due to a defunct schooling system and a poor socio economic background, which often originates from the Apartheid Era prior to 1994, the university has extensive support structures for students.

There is evidence at the university that many first year students are not able to achieve adequate results (HEMIS). In order to reduce this, many measures have been put in place by the university. For example, the SafeNet project has been established to investigate and identify students who are at risk of failing during their first year (Marnewick, 2011). These at-risk students are identified by their poor performance, where performance is monitored through tests and assignments. Students' assessments and assignment results are accumulated into a database over a given period of three to four weeks. Through the SafeNet project the data is collected and correlated. At-risk students are then identified as those students' who are failing to meet the minimum pass mark of 50%. These at-risk students are referred to learning centres, such as the Centre for Psychological Services and Career Development (PsyCad) (Johannesburg, 2012b), or the Division of Academic Development and Support (ADS) (Johannesburg, 2012a). Although the SafeNet project directly engages each at-risk student, no teaching-and-learning (Stetsenko, 2010) intervention is forthcoming. Therefore, although students are probed as to why they are not passing, and certain measures are taken to support the students, didactic processes are somewhat overlooked.

2.2 Teaching-and-learning Computer Programming

Teaching-and-learning is a term that has been coined because when combined, these words become the pathway through which cognitive, social, and effective development takes place (Stetsenko, 2002). This term originated from the scholar Vygotsky who expressed that "*Learning is not development; however, properly organized learning results in mental development and sets in motion a variety of developmental processes that would be impossible apart from learning*". To this end, teaching-and-learning should be a holistic process that allows students the opportunity to develop mental growth.

This is no different for teaching-and-learning computer programming as computer programming requires that skills, such as discipline, critical thinking and problem solving (Sprankle, 2009) be taught. Students, in turn need to cultivate these skills through practice. As students' practice they get an understanding of how to: solve problems; learn the syntax of a programming language; and design and construct computer programming solutions. To develop such skills students must be able to think critically and creatively (Sprankle, 2009, Kak, 2009). However, teaching students how to solve problems and how to acquire the necessary skills to become a skilled computer programmer is often a burdensome task (Kak, 2009, Ben-Ari, 1998); and students find it difficult to learn these skills, especially South African students (Koorsse, 2010a).

Computer programming as a subject and the pedagogies adopted to teach the subject, have not evolved much over the last five to ten years. In effect, the computer programming syllabus has consisted mainly of teaching students how to solve problems, in combination with teaching Java (Farrell, 2010), a programming language that is popular within industry. Although Java is a good language to make use of within industry, it is not an ideal language to teach novice computer programmers (Matthiasdottir, 2006). Furthermore, understanding difficult concepts is compounded by traditional teacher-centric pedagogies. Such pedagogies have been adopted to accommodate larger groups as student groups are between one hundred and two hundred students. A lecturer is seen as an expert and the primary source of knowledge (Wulf, 2005). Learning is a passive experience, and collaboration and discussion of computer programming concepts is kept to a minimum. Unfortunately, these teaching methods often do not succeed in imparting essential computer programming skills effectively (Robins, 2003).

At university level, teaching pedagogies mostly focus on lectures where student groups are large (ranging from one hundred to several hundred). These lectures are supplemented by dividing students into small groups of approximately twenty for laboratory (practical programming assignments are completed) and tutorial (revision of computer programming concepts) classes. This provides novice students with an opportunity to collaborate not only with peers but also with senior students and tutors (Daniels, 1996, Koorsse, 2010b, Vogts, 2008). The curriculum consists of a variety of Information Technology modules and computer programming is always a large contributor of such curricula. Computer programming modules are covered throughout first, second and third year levels (Johannesburg, 2013).

A closer look at the state of computer programming at the institution where this study was conducted revealed that failure rates are high for computer programming modules. For example, in 2012 nearly 60% of students failed after the first 6 months (semester one). These results indicate that learning computer programming for the first time is cognitively demanding (deRaadt, 2008). Given the demands of the discipline and given that South African students are underprepared for higher education, this inquiry investigates in what manner innovative pedagogical approaches could ease the burden and improve the potential of students. Although there has been extensive research into teaching-and-learning computer programming world-wide, information gaps do exist. This is especially true within a South African context.

2.3 An Alternative Pedagogical Approach to Teaching-and-learning

Although there are many solutions to teaching-and-learning, this article identifies social constructivism, collaboration and metacognition as alternative approaches that can be used to enhance teaching-and-learning.

2.4 Social Constructivism

Social constructivism is a didactic approach that allows learning to take place in a social, interactive and collaborative manner with the intention of students developing skills, such as reasoning, problem solving, the development of higher mental processes and metacognition (Kozulin, 2003). It is a paradigm that originated from scholars, such as Vygotsky, Piaget, Vico, Rorty and Bruner (Knowledgebase, 2012). Central to this theory is that teaching-and-learning is (Kozulin, 2003):

- An active process of constructing knowledge;
- A social process whereby students learn best in groups;
- Constructed on the experiences and hypotheses of the environment in which they learn;
- An experience that allows students to reflect and analyse solutions, also known as metacognition; and
- Shaped through the use of symbolic tools, such as language.

It therefore includes concepts, such as guided participation, appropriation, symbolic tools and collaboration amongst students (Kozulin, 2003). Parallels can be drawn between social constructivism and computer programming. For instance:

- Developing a computer programming solution is an active process of designing and constructing a solution;
- Computer programming solutions are developed through a process of discussion, where teams of individuals,

in collaboration with one another, solve problems and find solutions;
- Computer programming solutions are developed using symbolic tools, such as flowcharts, pseudo code and input-process-output (IPO) charts (Sprankle, 2009);
- The solution is created using a programming language, such as Java or C^{++}; and
- Solutions need to be tested and the student is therefore given a chance to reflect on their methods and results, also known as metacognition.

Social constructivism is grounded in other theories, such as collaboration, social development and metacognition. Collaboration and metacognition, in particular, can be applied to computer programming (Preston, 2005).

2.4.1 Collaboration

Collaboration is an effective pedagogy for computer programming (Preston, 2005) as computer programming is about solving problems and it is well known that students who solve problems in collaboration with one another learn from one another and teach others (Kozulin, 2003). This is due to the established criteria for tasks associated with computer programming, such as problem solving, creative thinking and critical thinking, being appropriate to collaboration-based pedagogy (Preston, 2005). Collaboration can be attributed to students (Preston, 2005, Preston, 2006b):
- Producing higher quality computer programs;
- Completing computer programs in a shorter amount of time;
- Understanding the computer programming process better;
- Enjoying the process of developing a computer program more; and
- Achieving improved pass rates for computer programming.

More than a century of research on collaborative learning has led to the observation that working together to achieve a common goal means that the overall result would be far greater than working alone (Preston, 2005). Although expressing ones thoughts is an effective method of learning, internal dialogue is also important for learning (Bransford, 2000).

2.4.2 Metacognition

Metacognition is an important strategy to make use of when developing computer programs as it is essential for a student to test and assess their computer programming solutions (Sprankle, 2009). In order to properly test and assess such solutions means that students need good metacognitive skills. In other words, students need to be able to “think about thinking”, also known as metacognition (Bransford, 2000). Metacognition can be defined as a student’s ability to plan, monitor and evaluate their cognitive processes (Thomas, 2001). It is an action that is deliberate and involves reflective thinking (Thomas, 2001). Metacognitive processes are imperative because, when a student develops a computer programming solution it is essential that the student is able to apply metacognitive thought, to ensure that the solution is effective, correct and succinct.

The proposed approach to teaching-and-learning computer programming is based on social constructivism, which includes key components, such as collaboration and metacognition. As discussed, such pedagogies provide an opportunity for students to “learn by doing”. Noted education scholars, such as Dewey and Piaget believed that children learn by doing and by thinking about what they do. Therefore, pedagogical approaches that encourage such learning provide an opportunity for student-centric learning, where knowledge is constructed by students and the lecturer is only a facilitator of learning rather than presenting information. Students grapple with problems and problems come before the teaching of the solution (Murray, 1998). Such teaching-and-learning encourages active participation where students collaborate with one another by comparing methods and solutions. The process of grappling with a problem and collaborating with peers enables students to engage in the learning process. When students engage in the learning process there is more chance that students will develop a deeper level of processing information, also known as higher order thinking skills (King, 2000). These skills provide an opportunity for students to apply, analyse, evaluate and create knowledge. Such learning develops more efficient problem solving skills (Xiaohui, 2006). The very skills that are needed by students to become compete computer programmers (deRaadt, 2008).

Research indicates that both Computer Science and Information Technology have earned a reputation for being difficult disciplines, not only world-wide (Fesakis, 2009) but also within South Africa (Koorsse, 2010a). It is well-known amongst researchers and lecturers alike that computer programming is a difficult subject to pass (Corney, 2010, deRaadt, 2005, Fincher, 2005, Robins, 2003). Pass rates are low (Havenga, 2009) and graphs for students learning computer programming often reflect bimodal distribution of marks (Robins, 2010). Given these statistics, many attempts to ease novice difficulties have been introduced. For example, in South Africa, tertiary institutions have:

- Introduced collaborative learning by dividing large groups up into approximately 20 students for laboratory and tutorial classes (Daniels, 1996);
- Introduced Scripting languages to teach computer programming instead of system programming languages, such as Java and C++ (Warren, 2001);
- Introduced Iconic programming notations, such as B# (developed at NMMU) to provide students with an opportunity to focus on problem solving skills instead of low-level programming details (Cilliers, 2005);
- Identified that the pass rate of grade 12 students that have taken Information Technology as a school subject is low and the level of proficiency is poor (Havenga, 2009);
- Identified motivation as an important aspect that influences student understanding (Koorsse, 2010b); and
- Identified that computer programming is difficult to such an extent that very few students are able to write simple programs that compile and run on completion of novice students' first computer programming course (Koorsse, 2010a, Vogts, 2008).

Although some researchers have made use of innovative pedagogical approaches within the South African context, the state of teaching-and-learning computer programming within the South African context is less than desirable (Havenga, 2009).

3. Research Methodology

3.1 The Approach

The study was conducted on a group of 36 first year students studying the National Diploma: Business Information Technology at the University of Johannesburg in 2012. The students were at-risk of failing their programming module namely, Development Software 1. A mixed methods approach was used to collect and analyse the data. Both quantitative and qualitative data pertaining to students' assessment results, a focus group interview and a questionnaire completed by the at-risk participants were used to determine whether the social constructivist pedagogy assisted students in overcoming their problems regarding computer programming.

By analysing student marks after they had completed their first major assessment in the form of a test, it became evident that almost 30% of students studying a computer programming module were at risk of failing the module. Consequently, students were invited to attend an extra programming lecture that was held every Friday for a total of six weeks. The extra lecture was facilitated by two lecturers and two tutors for a period of three hours per week. Each lecture was structured as follows:

- A short lecture was given;
- During the lecture, students were encouraged to ask questions, explain their reasoning to others, and respond to questions;
- The lecturer made a conscious effort to present problems and allow students to find solutions to such problems;
- Students were given additional problems to solve independently, but the students firstly discussed the problems in collaboration with one another, and secondly attempted to solve the problems independently;
- Lecturers and tutors played an active role and guided students as they collaborated with one another, and / or worked independently;
- Lecturers used scaffolding, such as flowcharts, pseudo code and tracing of computer programs to assist students in solving problems;
- Students were given an opportunity to reflect on how they had solved the problems, also known as metacognition (Kozulin, 2003); and
- Students were shown that they were now solving problems that they previously could not solve, which increased their level of confidence.

Table 1 below illustrates the extent to which the pedagogy overlapped with social constructivism pedagogy.

Table 1: The Approach Related To Social Constructivism

The Approach	**1**	**2**	**3**	**4**	**5**
During the lecture, students were encouraged to ask questions, explain their reasoning to others, and respond to questions;	√	√			
The lecturer made a conscious effort to present problems and allow students to find solutions to such problems;	√		√		
Students were given additional problems to solve independently, but the students firstly discussed the problems in collaboration with one another, and secondly attempted to solve the problems independently;	√	√	√		
Lecturers and tutors played an active role and guided students as they collaborated with one another, and / or worked independently;	√	√			
Lecturers used scaffolding, such as flowcharts, pseudocode and tracing of computer programs to assist students in solving problems;	√				√
Students were given an opportunity to reflect on how they had solved the problems, also known as metacognition;				√	
Students were shown that they were now solving problems that they previously could not solve, which increased their level of confidence.				√	
Social constructivism characteristics: **1 – active process; 2 – social process; 3 – constructed on experiences; 4 – metacognition; 5 – use of symbolic tools**					

4. Findings and Discussion

The findings relate to the quantitative and qualitative data collected, namely at-risk students assessment results, the focus group interview results, and the questionnaire completed by the at-risk students.

4.1 Student Assessment Results

Once the students had completed their second assessment, quantitative data (student marks) were assimilated, where the following comparisons were made:

- Equate the two assessments of the at-risk students who did attend additional lectures to determine whether the second assessment was an improvement over the first; and
- Determine whether additional lectures had a positive effect by comparing the second assessment mark of the at-risk students who did attend the extra lectures against the at-risk students who did not attend the extra lectures.

Figure 1 below illustrates that at-risk students' who attended the additional lecture, participated in class and completed the additional problem solving discussions, improved their second assessment result by an average of 11%. Each at-risk student's first assessment result (black) and their second assessment result (grey) can be seen. Only three students' results did not improve. Conversely, Figure 2 below illustrates assessment results of at-risk students who did not attend additional lectures. Interestingly, their second assessment results fell by 7%, which resulted in these students' performing poorly. Only four students' results improved and three out of the four were students who were repeating the course for the second or third time.

Therefore, the average assessment results for at-risk students who attended additional lectures were 41% (compared with their first assessment result of 30%). The average assessment results for at-risk students who did not attend additional lectures were 23%. However, although this quantitative data indicates improved overall results for at-risk students' attending additional lectures, this improvement may be attributed to the fact that lectures were simply repeated. Consequently, the study also included qualitative data (focus group interview) to determine whether it was the social constructivist pedagogy that positively influenced students' results.

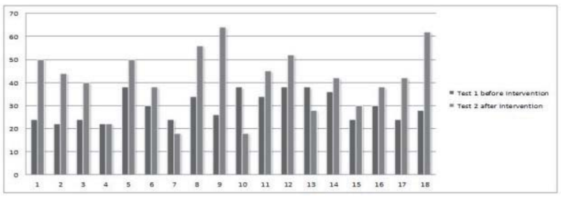

Figure 1: Assessment Results for At-risk Students - Did Attend

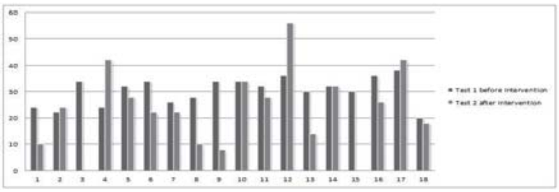

Figure 2: Assessment Results for At-risk Students – Did Not Attend

4.2 Focus Group Interview Results

Of the eighteen at-risk students who attended the extra programming lecture, nine volunteered to participate in a focus group interview to further explore their programming experiences. Open coding was used to analyse data collected from the focus group interview. Similar words and statements were highlighted and then labeled collectively. Once comparisons were made they were then grouped and categorized, as seen in Table 2 below.

Table 2: Final Theme Categories for the Focus Group

Student centered environment
Collaboration
Comfort levels
Time concerns

When students were asked by the lecturer what they either enjoyed or did not enjoy about the extra weekly lesson, the majority of students expressed that they enjoyed working in smaller groups with fellow students who were also struggling with programming. Students, however, would have preferred that these classes were held over a longer period of time and felt that six weeks was too short, as highlighted below.

Student 2:

" I liked working in groups, small groups; it is easier to explain what you understand with people you
"I know. What I did not enjoy, I think the time because I think we had too little time…"

Student 3: :

" The thing I enjoy a lot is working in groups and sharing ideas with other students…"

When students were asked how the extra lesson helped them to learn programming concepts they responded and said that the manner in which the lecturer allowed students to discuss and collaborate with the lecturer and other students when concepts were being explained made all the difference. This was especially true when it came to difficult concepts that students were struggling with. The students responded in the following manner:

Student 7:

"So when I come to the classes most of the time I just sit and listen and maybe like try to memorize so on Friday classes I was able to understand and know what was going on."

Student 9:

"The Friday class it was like revision to me …"

When students were asked if they preferred being in a class with the same type students who were also at risk of not getting a semester mark, (a mark that is calculated by adding all assessment results together. This mark serves as the determining factor regarding whether the student is allowed access to write an examination) or whether they would prefer to be in a mixed group they responded that they preferred being in a group with the same level of programming students. The responses were:

Student 2:

"I think we should remain in a group of similar students because it's small and compact and I feel that we are going somewhere."

Student 4:

"For me I think being in a group of same interests is better than being with good students, because good students can insult us and it may be annoying for us because we wouldn't know how to open our hearts like we did."

Student 5:

"Being in a small group it helped so much because being with the students with high marks the lecturer moves faster.'

From the focus group interview it emerged that students benefited from:

- Collaborating ideas and problems;
- Being part of a smaller group where active and interactive learning took place; and
- Being given ample opportunity to practice a variety of problems.

The above-mentioned benefits directly relate to the characteristics of social constructivism, as seen in Table 1 above. Additionally, these responses also relate to the manner in which they were taught. However, although the proposed pedagogy made a difference to students it was also important to further verify the manner in which the teaching-and-learning pedagogy had impacted the at-risk students. The students were asked to complete a questionnaire that directly related to the manner in which they were taught.

4.3 Questionnaire Results

Additionally, the at-risk students completed a questionnaire to further determine which aspects contributed to the extra programming class being a success. When students were asked whether there was a difference in the way that the extra programming classes were taught compared to the way in which the normal theory classes were taught 93% said that there was a difference and 7% said that there was no difference. Students who said that there was a difference contributed this difference to: the individual attention given; the class being more interactive; a repeat of the normal theory class; students feeling that they could ask questions; and a smaller group. These statistics are illustrated in Figure 3 below.

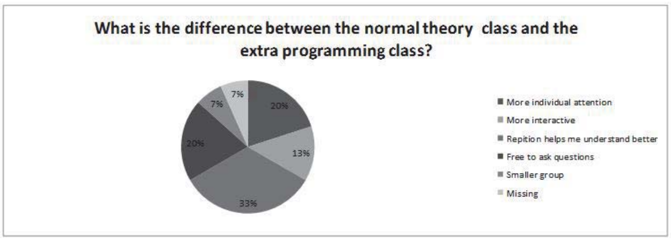

Figure 3: Differentiating Between Traditional and Proposed Pedagogy

Figure 4 below indicates how students felt they had benefited from the extra programming lectures. The students responded that: their marks had improved; programs were simplified; they could do the work on their own; and they understood concepts better.

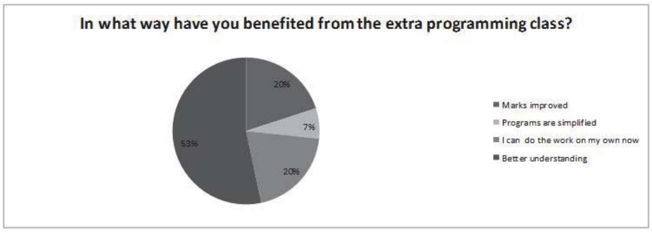

Figure 4: How Students Benefited

Figure 5 below indicates that students responded well to working in collaboration with one another. The students responded that they: could share ideas; enjoyed the discussions that took place; had assistance from other students; could see how other students answered the questions and in this way learn from them; and they felt that working in groups was more interactive.

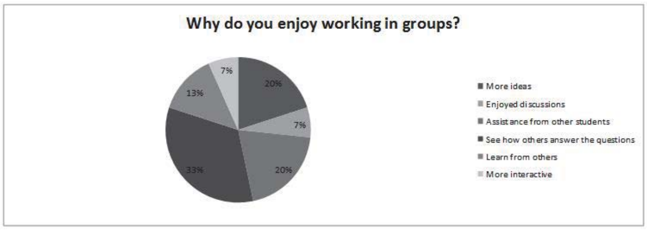

Figure 5: Reasons Why Students Enjoyed Working in Collaboration

Figure 6 below indicates that all of the students said that they enjoyed presenting their ideas on the board during the extra programming class and even if they did not get an opportunity to present their ideas, they still felt that it was a good idea as it helped them to: identify what they were doing wrong; understand visually; build up their self-confidence; be prepared before class; understand that they were all on the same level and they therefore did not need to feel intimidated; be more interactive; and be comfortable asking other students for help.

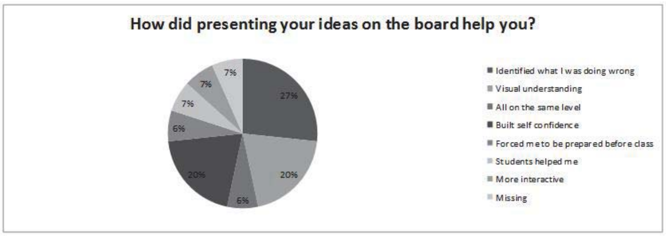

Figure 6: How Presenting on the Whiteboard Assisted Students

Finally, students were asked if they felt comfortable answering questions out loud in the extra programming classes. The response was 80% of the students said yes, 13% of the students said no and 7% did not respond. Figure 7 below indicates that the students who responded positively said that they felt comfortable because: students had a better understanding of the work; all the students were on the same level so they did not feel intimidated if they got the answer wrong; and the small class helped. Interestingly, the students who responded negatively either said that: they were too shy; or there were too many people and they did not feel comfortable expressing their thoughts.

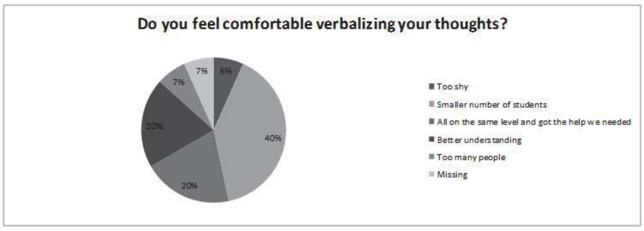

Figure 7: How Students Felt About Verbalizing Thoughts

4.4 Discussion

In using this pedagogical approach Table 3 below summarises the observations of the researchers regarding how the pedagogical approach assisted students and what problems they experienced. This table reflects computer programming design principles that can be used by educators to assist students in improving their computer programming skills.

Table 3: Observations of the Pedagogical Approach

How did students learn?	**Positive attributes of the learning**	**Negative attributes of the learning**
Collaboration	• Working with others that were on the same level • Sharing ideas • Assistance from other students • Learning from others	• Needed to spend more time collaborating
Small class group	• Comfort levels were elevated	• Needed more than just once a week working in small classes
Discussion and dialogue that took place during the lecture	• Students felt comfortable voicing their opinions • Built self confidence • More interactive • Forced to prepare before class	• Too shy to answer questions and raise an opinion
Repetition	• Repeating a lecture builds metacognitive skills as it allows the student an opportunity to assess what they understood when the lecture was first received as opposed to the repeated version	

The interviews allowed students to offer their opinions, which are subjective, but these reflected opinions run parallel with the characteristics associated with social constructivism. Given their opinions, in combination with an improvement in students' results (post assessment), which is a good indicator of success, it is therefore quite possible that the manner in which the at-risk students were taught, encouraged them to acquire the necessary skills needed to improve their computer programming skills. Additionally, as the pedagogical intervention was the only differentiator between the pre and post assessments, in combination with the fact that learning approaches are powerful determinants of success (deRaadt, 2005) it is most likely that students results improved because of the pedagogical intervention.

To summarize, the overall findings related to the assessment results comparison; the focus group interview; and the results of the student questionnaires, indicate that most students:

- Who attended additional lectures improved their semester mark, as opposed to at-risk students who did not attend such lectures;
- Learnt better in a student-centered environment, where they could collaborate with one another;
- Learnt to apply metacognitive thought as they were able to analyse their solutions; and
- Benefited from an environment that: was more interactive, encouraged discussion and the sharing of ideas; allowed them to see how fellow students solved problems; actively engaged in the lecture being given; gave them the confidence to work independently; and allowed them to identify where they were going wrong when solving a problem.

5. Concluding Remarks

This study has shown that adopting social constructivist pedagogy can play an important role in improving the problem solving and computer programming skills of at-risk students. This in turn, improved the pass rate for a computer programming course directed towards novice students. As this was a pilot study, it is envisaged that the pedagogy will be improved and extended to include all students, at the start of the course. It is hoped that by applying this type of teaching-and-learning, less students will be at risk of failing this course. This in turn, should equate to an improved pass rate for Development Software I, which means more employable students and a decrease in unemployed young adults.

Although this paper describes a move towards constructivist pedagogy, future research will focus on developing teaching-and-learning principles using a design-based research paradigm. These teaching-and-learning principles can then be used to inform the pedagogy for all novice computer programming students.

References

Ben-Ari, M. 1998. Constructivism in Computer Science Education. In: SIGSCE, Ed. SIGSCE'98, 1998 Atlanta GA, USA. ACM.
Boughey, C. 2009. Access and Success In Higher Education: Being In and of the University.
Boyer, N. R., Langevin,S.,Gaspar,A. 2008. Self Direction & Constructivism in Programming Education. In: SIGITE, Ed. SIGITE', 2008 Cincinnati, Ohio, USA. ACM.
Bransford, J. D., Brown, A.L., Cocking, R.R. 2000. How People Learn: Brain, Mind, Experience, and School: Expanded Edition. In: Practice, C. O. L. R. A. E. (Ed.). The National Academies.
Cilliers, C., Calitz, A., Greyling, J. 2005. The Effect Of Integrating an Iconic Programming Notation into CS1. In: ACM, Ed. ITICSE ' 05, 2005 Monte De Caparica, Portugal. ACM, 5.
Corney, M., Teague, D., Thomas, R.N. 2010. Engaging Students in Programming. In: Australian Computer Society, I., Ed. Twelfth Australasian Computing Education Conference (ACE2010), January 2010 2010 Brisbane, Australia. Australian Computer Society, Inc.
Daniels, M., Gal-Ezer, J., Sanders, I., Teague, G.J. 1996. Teaching Computer Science: Experiences from Four Continents. In: ACM, Ed. SIGCSE '96, 1996 Philadelphia, USA. ACM, 5.
DeRaadt, M. 2008. Teaching Programming Strategies Explicitly to Novice Programmers. Doctor Of Philosophy, University of Southern Queensland.
DeRaadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., Cutts, C., Fincher, S., Hamer, J., Haden, P., Petre, M., Robins, A., Simon, Sutton, K. 2005. Approaches to Learning in Computer Programming Students and their Effect on Success. University of Otago, New Zealand: ACM SIGCSE Special Projects Grant.
Farrell, J. 2010. Javatm Programming, Fifth Edition, Course Technology, Cengage Learning.
Fedderke, J. W., De Kadt, R., Luiz, J.M. 2000. Uneducating South Africa: The Failure to Address the 1910-1993 Legacy. International Review of Education, 46, 257-281.
Fesakis, G., Kiriaki, S. 2009. Influence of the Familiarization with "Scratch" on Future Teachers' Opinions and Attitudes about Programming and ICT in Education. In: ITICSE'09, 2009 Paris, France. ACM.
Fincher, S., Baker, B., Box, I., Cutts, Q., Deraadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Simon, Robins, A., Sutton, K., Tolhurst, D., Tutty, J. 2005. Programmed To Succeed? A Multi-National, Multi-Institutional Study of Introductory Programming Courses. Univeristy Of Kent.
Havenga, M., Mentz, E. 2009. The School Subject Information Technology: A South African Perspective. In: ACM, Ed. SACLA '09, 2009 South Africa. South Africa: ACM, 7.
Hungi, N., Thuku, F.W. 2010. Variations In Reading Achievement Across 14 Southern African School Systems: Which Factors Matter? International Review of Education, 56, 63-101.
Jansen, J. 2012. I Would Seriously Consider Not Sending My Child to School in SA [Online]. Moneyweb. Available: Http://Www.Moneyweb.Co.Za/Mw/View/Mw/En/Page292681?Oid=562851&Sn=2009%20detail [Accessed 24 February 2012].
Johannesburg, U. O. 2012a. ADS [Online]. Available: Http://Www.Uj.Ac.Za/En/Ujsearch/Pages/Searchresult.Aspx/Results.Aspx?K=Ads&S=All [Accessed Jauary 2012].
Johannesburg, U. O. 2012b. Psycad [Online]. Available: Http://Www.Uj.Ac.Za/En/Ujsearch/Pages/Searchresult.Aspx/Results.Aspx?K=Psycad&S=All [Accessed January 2012].
Johannesburg, U. O. 2013. Curriculum Guidelines For The Faculty Of Management. In: Johannesburg, U. O. (Ed.). Johannesburg: University Of Johannesburg.
Kak, A. 2009. Teaching Programming. Available: Https://Engineering.Purdue.Edu/Kak/Teachingprogramming.Pdf [Accessed November 2010].
King, F. J., Goodson, L., Rohani, F. 2000. Higher Order Thinking Skills. Assessment Evaluation - Educational Services Program.
Knowledgebase, L. T. 2012. Learning-Theories.Com. Available: Http://Www.Learning-Theories.Com/ [Accessed September 2011].
Koorsse, M., Calitz, A.P., Cilliers, C. 2010a. Programming in South African Schools: The Inside Story. SACLA2010. ACM.
Koorsse, M., Cilliers, C., Calitz, A.P. 2010b. Motivation and Learning Preferences of Information Technology Students in South African Secondary Schools. In: ACM, Ed. SAICSIT '10, 2010b Bela Bela, South Africa. ACM.

Kozulin, A., Gindis, B., Ageyev, Vladimir S., Miller, Suzanne M. (Ed.) 2003. Vygotsky's Educational Theory In Cultural Context: Cambridge.

Marnewick, C. 2011. The Mystery Of Students Selection: Are There Any Selection Criteria. Educational Studies, 38, 15.

Matthiasdottir, A. 2006. How to Teach Programming Languages to Novice Students? Lecturing or Not? In: International Conference on Computer Systems And Technologies - COMPSYSTECH'06, 2006.

Murray, H., Olivier, A., Human, P. 1998. Learning Through Problem Solving. In: Newstead, A. O. K., Ed. Proceedings Of The Twenty-Second International Conference for the Psychology of Mathematics Education, 1998 Stellenbosch, South Africa. 17.

Preston, D. 2005. Pair Programming As a Model Of Collaborative Learning: A Review of the Research. CCSC: Central Plains Conference. Consortium for Computing Sciences in Colleges.

Preston, D. 2005. Adapting Pair Programming Pedagogy for Use in Computer Literacy Courses. In: CCSXC:Mid-South Conference, 2006a. JCSC.

Preston, D. 2006b. Using Collaborative Learning Research to Enhance Pair Programming Pedagogy. ACM SIGITE, 3.

Robins, A. 2010. Learning Edge Momentum: A New Account of Outcomes in CS1. Computer Science Educational 20.

Robins, A., Rountree, J., Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion. Computer Science Educational Journal, 13. 137-172.

Sprankle, M. H., J. 2009. Problem Solving & Programming Concepts, New Jersey, Pearson Education.

Stetsenko, A. 2010. Teaching-Learning And Development as Activist Projects of Historical Becoming: Expanding Vygotsky's Approach to Pedagogy. Pedagogies: An International Journal, 5, 6-16.

Stetsenko, A., Arievitch, I. 2002. Teaching, Learning, And Development: A Post-Vygotskian Perspective.

Thomas, G. P. 2001. Toward Effective Computer Use in High School Science Education: Where to From Here? Education and Information Technologies, 6.

Van Zyl, A. 2012. First Year Student Research: Faculty of Management. University of Johannesburg.

Vogts, D., Calitz, A., Greyling, J. 2008. Comparison of the Effects of Professional and Pedagogical Program Development Environments on Novice Programmers. In: ACM, Ed. SAICSIT '08, 2008 Wilderness, South Africa. ACM.

Warren, P. 2001. Teaching Programming Using Scripting Languages. In: CCSC, Ed. Consortium for Computing In Small Colleges (CCSC): Southeastern Conference, 2001. JCSC, 12.

Wulf, T. 2005. Constructivist Approaches for Teaching Computer Programming. In: SIGITE'05, 2005 New Jersey, USA. ACM, 4.

Xiaohui, H. 2006. Improving Teaching in Computer Programming by Adopting Student-Centered Learning Strategies. The China Papers.