An Investigation Into The Relationship Between The Level Of Cognitive Maturity And The Types Of Errors Made By Students In A Computer Programming Course

S. Ranjeeth, (E-mail: ranjeeths@ukzn.ac.za), University of KwaZulu-Natal, South Africa
Ramu Naidoo, (E-mail: naidoo@dut.ac.za0), Durban University of Technology, South Africa

ABSTRACT

Computer Programming forms the basis from which most students in the IS/IT discipline launch themselves into further endeavors in the discipline. However, statistical analysis of students' performances in programming related assessments tasks reveals that the mastery of computer programming skills is not easily acquired. This assertion is supported by the reports of high failure rates in programming courses at several academic institutes. This trend is also confirmed at the University of KwaZulu-Natal (UKZN) where programming related assessments have resulted in failure rates as high as 50%. The study encompasses a hybrid model consisting of a semi-structured interview and an error analysis of student responses in a programming examination. Students were grouped into cognitive level 1, 2 and 3. A qualitative theoretical framework of Piaget and error classification using Naidoo (1996) The results indicate that cognitive group 1 exhibited the most structural errors whilst the cognitive group 3 exhibited the least structural errors. .

INTRODUCTION

Since the inception of the concept of software development, the mastery of computer programming seems to have been tagged as perpetually problematic. This assertion can be substantiated by sentiments echoed by Edsger Dijkstra in his ACM Turing Award Lecture, "The Humble Programmer" where he said:

…as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, now we have gigantic computers, programming has become an equally gigantic problem" (Dijkstra,1972).

Pea and Kurland (1984) cite numerous studies that allude to the deep misunderstanding of programming related concepts by adult novice programmers as a result of a lack of pedagogical theory on computer programming. This is certainly a concern for academics as well as students aspiring to become experts in the field of IT. Efforts to unravel the mystery behind this dilemma have been hampered due to the dynamism inherent in computer programming. A constant trait of computer programming is the ever changing technological infrastructure as well as the significant paradigm shifts in terms of programming strategy [13, 21 & 23]. It is around this fluctuating environment that research into the programming ability of students is often conducted. This paper does not attempt to factor out the many variables that may influence a study into the programming abilities of students. However, in order to gain an insight into issues that influence the learning of computer programming, an investigation of the relationship between the cognitive maturity levels of students and their understanding of basic programming related concepts is undertaken.

THEORY OF COGNITIVE MATURITY

The generic definition of the concept of intelligence alludes to traits such as the ability to deal with abstractions, the ability to solve problems and the ability to learn (Slavin, 1991: 227). A commonly used instrument to measure intelligence levels is the Intelligence Quotient (IQ) test developed by Alfred Binet in the early 1900s.

From a Piagetian perspective, the highest level of cognitive development is defined by the Formal Operations stage of development. This stage embodies the ability to deal with abstractions, form hypotheses, solve problems and engage in mental manipulations. The resemblance to the concept of intelligence as defined above is significantly apparent.

Hence an instrument to ascertain the intelligence/ cognitive maturity levels required to excel in programming should incorporate elements of the IQ test, but also have a strong focus on programming related concepts. Lawson (1983) observed that a precondition to formal operations is to understand bi-conditional reasoning, "if and only if" logic. In terms of procedural programming, this is significant since understanding of bi-conditional reasoning is pivotal in terms of learning with respect to procedural programming (White & Sivitanides, 2002). Burton and Bruhn (2003) argued that the mastery of the basic tenets of procedural programming is vital to ensure success with respect to implementation in an object oriented environment. This assertion makes the relevance of bi-conditional reasoning independent of the paradigm of development.

It is within this framework of reasoning that an instrument to measure the cognitive maturity of students to handle the rigors of learning to program was assembled. Previous research efforts have not resulted in many definitive claims to the discovery of cognitive pre-requisites for learning programming. However, many factors have been mentioned as possible contributors. These include mathematical ability, memory capacity as well as logical, conditional and temporal reasoning skills. This paper focuses on the analogical realm and the ability to engage in problem solving by invoking the abstract domain. Previous research has shown that experts think about problem solutions abstractly (Petre and Blackwell, 1997; Adelson and Soloway, 1985). The purpose of the instrument is to measure the impact that logical and abstract problem solving ability has on programming competence levels.

The instrument consisted of abstract reasoning, propositional logic and analytical testing as demonstrated by the samples shown below:

Consider the following rule about a set of cards that have letters on one side and numbers on the other side.

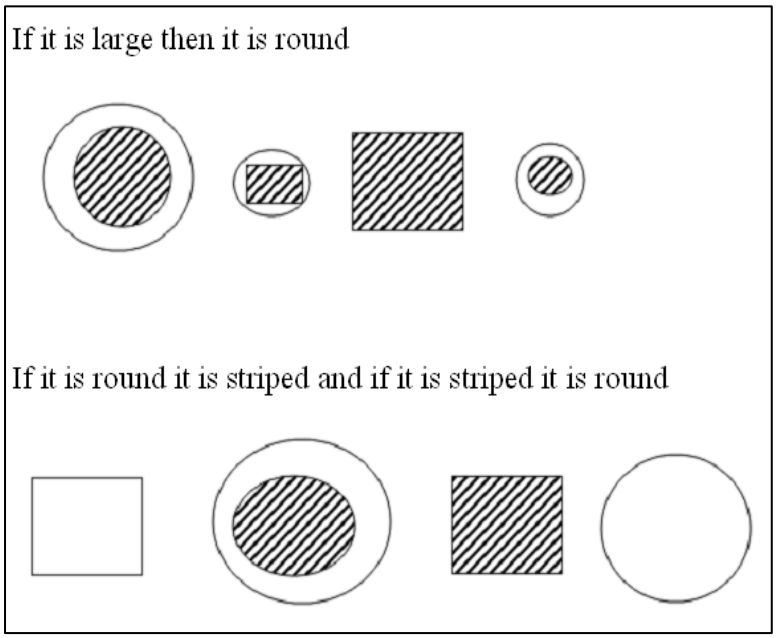
"If a card has a vowel on one side, then it has an even number on the other side."

Look at the cards below and determine which cards need to be turned over to verify if this rule is true?

E	F	4	7

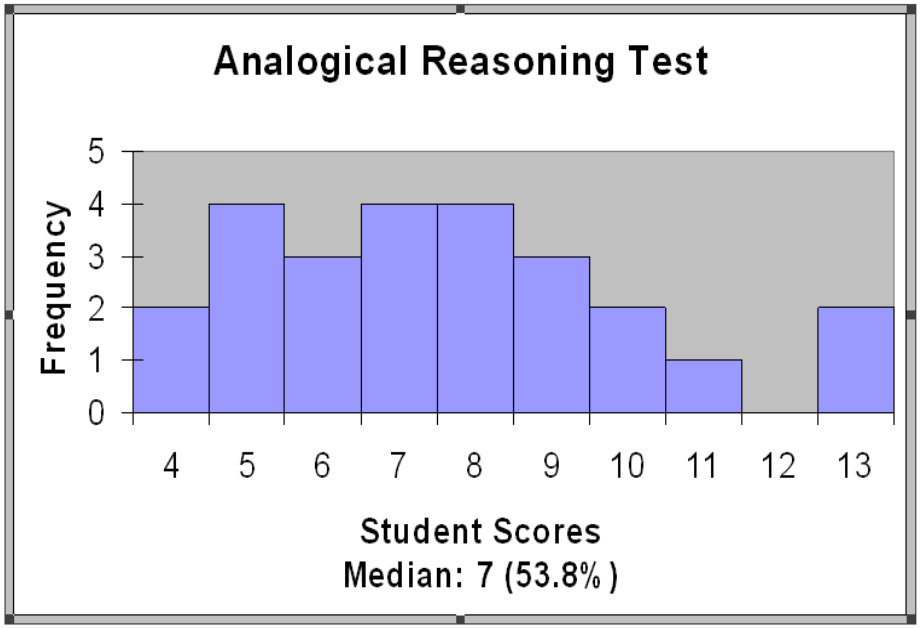
Write down your response to this question.

Wason and Laird-Johnson, 1972



Piburn 1989 – Key to the Propositional Logic Test

The test, consisting of 13 questions, was administered to a group of 25 students from an introductory programming class at the University of KwaZulu-Natal. An analysis of responses is manifested in the following graphical representation:



A standard deviation of 2 indicates that the median value is a good global approximation of the performance of respondents in this test. An initial summation will reveal the significant observation that 36% of the sample taken did not exhibit adequate bi-conditional reasoning skills that would be a pre-requisite to ensure mastery of programming related concepts. The bias of the sample from a race and age perspective seems to have been negligible. The gender factor was in accordance with data from historical research (as cited in Cukier, Short and Devine, 2002) where there is a trend that suggests that female involvement in technologically oriented courses is significantly lower than males. However, variable influence manifested itself in the form of previous exposure to computer programming. Sixty four percent of those respondents that scored above 50% have indicated that they have had previous exposure to computer programming at varying intensity levels. This observation is in accordance with assertions by cognitive theorists that prior experiences and knowledge is required to facilitate assimilation and understanding of new information. These theories are manifested in the constructivist epistemologies of learning as advocated by Piaget and further entrenched by the philosophies of Schema Theory as proposed by Asubel (précised in Slavin, 1991:227) Generative Learning Theory proposed by Wittrock (précised in Slavin, 1991:268) and Davis's Frame Theory (1968). The conclusion here is that these students were able to elicit an innate, reflexive response to most of the questions on bi-conditional reasoning. However, it was also significant to observe that 72% of the students were not able to provide a correct response to the question that required the invocation of abstract reasoning skills.

The developmental cognitive science perspective is based on the prediction that "…a student's attainable level of programming skill may be constrained by cognitive abilities required in programming" (Pea and Kurland, 1984). The ability to deal with abstractions and engage in mental manipulations, referred to as the formal operations stage is the highest Piagetian stage of cognitive development. Research has shown that a majority of college students fail at many formal operational tasks (Schwebel, 1975). White and Sivitanides (2002) cite various studies that report on the definitive influence of cognitive developmental level on one's ability to learn programming. The conclusion is that people who have reached Piaget's formal operational stage would have the tools needed to understand programming. However, Barry Kurtz (1980) added a more detailed classification by introducing the early and late formal stages of development. The two students who were able to answer the question on propositional logic correctly went on to score full marks on the test. The cognitive development of these students could be confidently classified as late formal. However, in terms of overall performance, students bordering on a score of 75% and above have also been classified as late formal. This constituted 20% of the sample. The remainder of the sample would qualify to be either early formal or late concrete. This constituted 80% of the sample. A possible conclusion here is that a high proportion of the students in this study have either marginal, or severely lacking requisite cognitive maturity levels to ensure mastery of programming concepts.

ERROR ANALYSIS

In order to establish the misconceptions that may underlie attempts at mastery of computer programming, an analysis of errors made by students of programming need to be undertaken. However, there is a paucity of research material on the analysis of errors made by novice programmers. Although cognitive scientists have ventured a unanimous assertion that students with higher levels of cognitive maturity are better equipped to handle the challenges of learning to program, there have not been many efforts to correlate the type of errors made by students and their levels of cognitive maturity. This kind of analysis would be vital in providing a causal basis for necessitating higher cognitive attributes of students in programming courses.

In order to achieve such an error analysis, an adapted version of the scheme described by R.Naidoo(1996) is used to analyse student responses in a programming exam. The error categories presented here in increasing order of severity:

- Arbitrary Error – subject behaves arbitrarily and failed to take into account the constraints laid down. A programming related adaptation of this would be the identification of low level errors such as syntactically driven errors or inappropriate use of syntactic constructs – described by Schneiderman & Mayer (1979) as "…precise, detailed and arbitrary knowledge about how the constructs in a particular language must be implemented"

- Executive Error – subject failed to carry out manipulations though the principles involved may have been understood. A programming related adaptation of this would be manifest in solutions that may generate run-time errors or the generation of incorrect output due to semantic incorrectness. Schneiderman & Mayer described semantic knowledge to include low level concepts such as understanding the concept of assigning a value to a variable, summing the contents of an array, and understanding of high level concepts such as searching and sorting of data values.

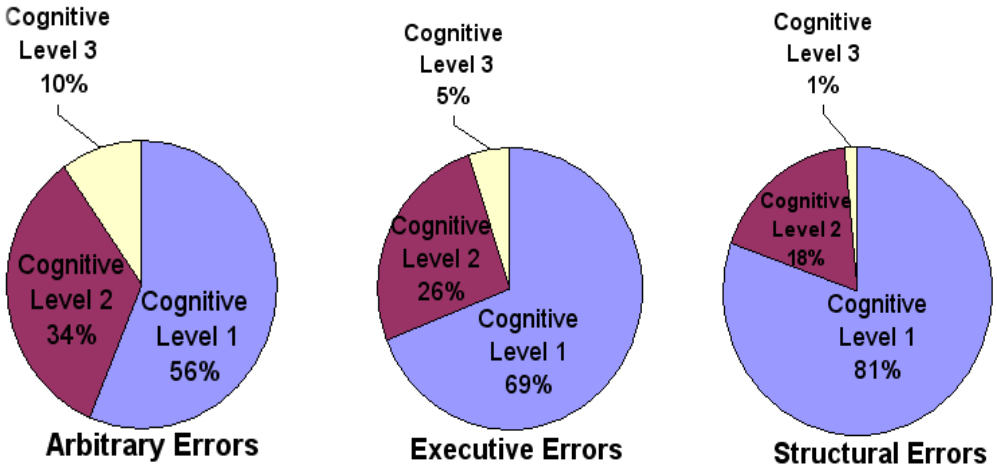
- Structural Errors – subject failed to appreciate the relationship involved in the problem or to grasp some principle essential to the solution. A programming related adaptation would be the inability to comprehend the problem domain or the inability to create a programming model to represent the problem domain.

In order to do a statistical analysis of errors made, the following classifications were used. Students' performance in relation to various aspects of programming was assessed on a five point scale. However, each of the aspects were further analysed in terms of the students' perceived levels of cognitive maturity using the Piagetian framework (Late concrete – Level1; Early Formal – Level2; Late Formal – Level3).

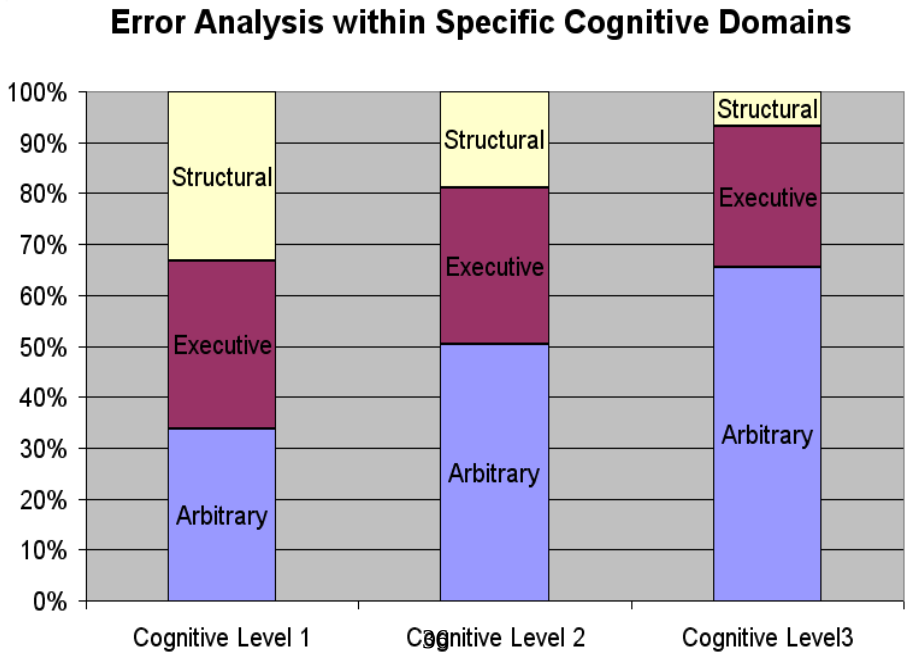
Description	Cognitive Maturity Level	Error Frequency		
		Arbitrary	Executive	Structural
Flowchart Implementation	1	13	17	4
	2	7	5	2
	3	5	2	0
Algorithmic Manipulation	1	12	7	2
	2	4	1	0
	3	3	0	0
Conditional Structure Implementation	1	18	15	9
	2	7	3	1
	3	1	2	1
Looping Structure Implementation	1	13	11	21
	2	12	8	4
	3	2	0	0
Array Manipulation	1	19	23	25
	2	11	9	3
	3	2	2	0
User defined data structure construction	1	17	20	23
	2	15	7	6
	3	4	2	1
Parameter Passing/ Modular Programming	1	18	25	27
	2	11	7	9
	3	2	0	0

DISCUSSION

Averages of errors in each category were used to create a graphical representation that reflected an adequate summary of the data presented above. The graphical representations depicted a trend that highlights cognitive maturity level as a significant contributory factor to the overall set of errors detected.



The relatively low frequency counts for Executive and Structural errors in the categories of Flowchart and Algorithm analysis can be attributed to the fact that these questions did not incorporate elements of abstract reasoning. However, questions that required an invocation of problem solving skills, incorporating a high level of abstract reasoning, resulted in a trend that could be surmised as follows: students with a higher level of cognitive maturity (Levels two and three) have a significantly lower incidence of Executive and Structural errors. There is a strong negative correlation between error counts and the levels of cognitive maturity. Another significant observation with respect to error trends within each of the cognitive groups is revealed by the following graphical representation.



The percentage contribution of Structural errors diminishes significantly with each transition from a lower cognitive level to a higher one. This is indicative of a progressively higher level of problem domain comprehension with each successive cognitive level. A possible misnomer in the graph above is the relative equivalence of the different error types as observed in the case of Cognitive Level One students. However, this can be explained by the difficulty encountered in trying to establish a definitive distinction between Executive and Structural errors. A possible explanation for the Executive errors is that such errors could be attributed to a lack of understanding of the problem domain, in which case, these could well be classified as Structural. In some instances, the severe nature of an Arbitrary error could well have diluted the case for the error to be classified as a Structural error, hence resulting in a possible incorrect classification. These anomalies were however not prevalent in the samples taken from the higher cognitive categories. An explanation for Executive type errors would suggest the prevalence of a "semantic gap" which has been described by Blackwell (1996) as a lack of coherence between the programmer's conceptual model of what their program should do, and the computational model of the program itself. The data analyses would seem to suggest that this "semantic gap" manifests itself a lot more significantly for respondents from the lower cognitive grouping. This lack of semantic mobility would have an error generating cascading effect that would be detrimental in facilitating any kind of meaningful analysis of the problem situation. In order to demonstrate this assertion, a data structure declaration is shown below:

```
Structure Sales
        Dim Name As String
        Dim Sales() As Integer
End Structure
Dim Employees(9) As Sales
```

In order to become effective users of the data structure, students had to conceptualise the structure and become comfortable with the idea of engaging in subsequent manipulation of it. However, this conceptualisation process would be constrained by the student's level of understanding of fundamental data structures such as arrays and structures.

Richard Mayer (1981) writes that understanding is "…the ability to use learned information in problem-solving that are different from what was explicitly taught". This is manifested in the user's ability to transfer learning to new situations. Hence, a higher level of cognitive capability would facilitate transfer of understanding of fundamental concepts onto the newly encountered problem domain. A process referred to as "assimilation" according to the Piagetian framework. In the case in discussion, a reliance on knowledge of arrays and structures ("learned information") would be required to facilitate comprehension of an array of structures ("new situation"). In the case of students with poor assimilation qualities, the consequence would be a prevalence of Structural errors which in turn would have a "knock on effect" resulting in a high incidence of Executive errors. This assertion is corroborated by student responses to a task that required data input into the structure defined above. A solution to this task would resemble the following modularised code segment:

```
Sub GetData(ByVal X() As Sales)
 Dim i As Integer
 For i = 0 To X.GetUpperBound(0)
   X(i).Name = InputBox("Enter Name of Employee")
   X(i).TotalSales = InputBox("Enter Amount of sales")
 Next
End Sub
```

An analysis of student responses revealed that students failed to carry out manipulations in order to facilitate proper data capture. This is a case of an Executive error that has come to fruition as a consequence of a lack of comprehension of the structure used to model the problem domain. By definition, this becomes an instantiation of a situation where a Structural error has become the catalyst for a possible series of consequential Executive errors.

CONCLUSION

The results and analyses presented above provide evidence that students with a lower level of cognitive maturity tend to engage in surface/artificial learning of computer programming concepts. This is substantiated by an analysis of the severity of the errors incurred when writing programs. This assertion is supported by a recent phenomenographic study, (Booth, 2001) where it was found that students, who approached learning to program as learning to code in a language or as passing the course, exhibited a surface approach to learning. This kind of practice would not be condoned by the academic community. However, the current pedagogical examining style as espoused by Bloom's Taxonomy requires that assessment tasks be categorised to cater for the varying cognitive levels. These range in a continuum from simple recall skills to higher order analytical skills. It is within the confines of this taxonomy, that students with a surface knowledge of programming concepts may achieve their objective of passing a programming course. This assertion is based on the observation that the severity of Structural errors is diluted by the presence of tasks that require only mechanical manipulation or understanding of basic concepts. These tasks could only possibly generate Executive errors, thereby obscuring the Structural inadequacies the student may possess. This argument presented here illustrates the potential of current teaching strategies for creating a situation that allows students, with unsustainable abilities as programmers, to be deceived into classifying themselves as experts.

RECOMMENDATIONS

"Teaching is a scholarly activity and a life-long learning process with no single method or pedagogy that is always most effective" (Ali, 2005). However an acknowledged objective of the academic fraternity is to produce graduates who have established mastery over subject matter, thereby enhancing their creative and critical thinking abilities. In order to achieve this objective in an analogically demanding discipline such as computer programming, there needs to be an adoption of innovative strategies in order to counter the seemingly perpetual rate of failure.

There is a need to invest in adopting a diversified strategy to teach computer programming. This should be based on the level of cognitive maturity of the student as opposed to the traditional teaching strategies that is based on the assumption that all students have an equivalent cognitive capacity to handle learning of new material. Teaching strategies at various levels of intensity could then be explored in order to compensate for the lower cognitive learners. This approach could enhance the possibility of deeper learning traits being manifested across a wider range of computer programming students.

Although these recommendations have been ventured on the basis of a pilot study and a subset of the Piaget Logical Tests, it is suggested that such a study be replicated on a larger scale with a more comprehensive test of a subject's analogical capability.

REFERENCES

1. Adelson B. and Soloway E. (1985) The Role of domain experience in software design IEEE, *Transactions on Software Engineering*, SE-11 (11), 1351-1360.
2. Ali S. (2005) *Effective Teaching Pedagogies for Undergraduate Computer Science Mathematics and Computer Education* 39(3) pp. 243.
3. Blackwell A. (1996) Metacognitive Theories of Visual Programming: What do we think we are doing? IEEE Symposium on Visual Languages, 1996, pp.240.
4. Bergin S. and Reilly R. (2005) Programming: Factors that influence Success, Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, pp: 411-415.
5. Brooks F. (1987) No Silver Bullet: Essence and Accidents of Software Engineering. *Computer.*

6. Booth S., (2001) Learning to program as entering the datalogical culture: A phenemenographic exploration. In the 9th European Conference for Research on Learning and Instruction, Fribourg, Switzerland.
7. Burton P. and Bruhn R. (2003) Teaching Programming in the OOP Era, *SIGCSE Bulletin*, ACM Publishers, 35(2) pp. 111-114.
8. Byrne P. and Lyons G. (2001) The Effect of Student Attributes on Success in Programming, in SIGCSE Bulletin – inroads-Proceedings of ITiCSE 2001, Vol 6, No. 1, pp.49-52, ACM Press.
9. Cukier W., Short D., and Devine I., 2002, Gender and Information Technology: Implications and definitions, *Journal of Information Systems Education*, Vol 13(1), pp.:7-15.
10. Davis, R.B. (1984). *Learning Mathematics. The Cognitive Science Approach to Mathematics Education*. Kent:Croom-Helm Ltd.
11. Dijkstra, E.W. (1972) The Humble Programmer. *Communications of the ACM*, Vol. 15 (10), pp. 859-866, 1972.
12. Donaldson, M. (1963). *A study of children's thinking*. London: Tavistok Publications.
13. Dobosiewicz W., Mahmoud Q., and Swayne D. (2004) Redesigning Introductory Computer Programming with HTML, Javascript and Java, Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, pp. 120-124.
14. Kurtz B. (1980) Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class Proceedings of the 11th SIGCSE Technical Symposium on Computer Science Education pp. 110 – 117 , ISSN: 0097-8418.
15. Lawson A.E. (1983) The Acquisition of formal Operational Schemata during Adolescence: The Role of the Biconditional, *Journal of research in Science Teaching*, 20(4), pp. 347-356.
16. Mayer R., (1981) The Psychology of How Novices Learn Computer Programming *ACM Computing Surveys Archive*, 13(1), pp. 121-141, ISSN: 0360-0300.
17. Naidoo R. (1998) Errors made by First Year Technikon Students in DE. PME International Conference, Stellenbosch, South Africa.
18. Pea D.R. and Kurland M. (1984) On the Cognitive effects of learning Computer Programming, *New Ideas in Psychology*, 2(2), pp.137-168.
19. Petre M. and Blackwell A. (1997) A glimpse of expert programmers' mental imagery Paper presented at the 7th workshop on Empirical studies of programmers, pp. 109-123, ISBN: 0-89791-992-0.
20. Piburn M. (1989) Key to the Propositional Logic Test Unpublished document. Accessed via http://www.cs.utexas.edu/users/almstrum/PLT/key.pdf.
21. Mitchell W., T. (2000) A Paradigm shift to OOP has occurred…implementation to follow, Proceedings of the fourteenth annual consortium on Small Colleges Southeastern conference, Published in the Consortium for Computing Sciences in Colleges archive, pp.94-105.
22. Slavin R. (1991) *Educational Psychology: Theory into Practice* (3rd Edition), Prentice Hall, ISBN: 0-13-237751-9.
23. Selker T. (1996) New Paradigms for using Computers, *Communications of the ACM*, 39(8), pp. 60-69.
24. Schwebel M. (1975) Formal operations in first year college students Published in the *Journal of Psychology*, 91, pp. 133-141.
25. Shneiderman B. and Mayer R. (1979) Syntactic/semantic interactions in programmer behavior: A model and some experimental results Published in the *International Journal of Computer and Information Sciences*, 8, pp. 219-238.
26. Trott P. (1997) Programming Languages: past, present, and future: sixteen prominent computer scientists assess our field, *ACM SIGPLAN Notices Archive*, 32(1), pp. 14-57.
27. Wason P. and Johnson-Laird P. (1972) *Psychology of Reasoning: Structure and Content*. Cambridge: Harvard University Press
28. White G. and Sivitanides M. (2002) A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics, *Journal of Information Systems Education*, 13(1) pp. 59-66.

NOTES