INTRINSIC AND EXTRINSIC FACTORS AS PREDICTORS OF SELF EFFICACY AND ACHIEVEMENT IN PROGRAMMING AMONG COMPUTER SCIENCE UNDERGRADUATES IN SOUTH-WESTERN NIGERIA

OWOLABI JOSIAH

B.Sc. Ed (Lagos), M.Sc. Mathematics (Lagos), PGD Computer Science (Lagos), M.ScComputer Science (Lagos), M.Ed. Educational Evaluation (Ibadan)

Matric No: 131284

A Thesis in the International Centre for Educational Evaluation (ICEE) Submitted to the Institute of Education in Partial Fulfilment of the Requirement for the degree of DOCTOR OF PHILOSOPHY of the University of Ibadan, Ibadan, Nigeria.

2014

ABSTRACT

Java programming language is recent, dynamic and relevant in contemporary organisations. However, literature shows that learning computer programming using Java poses problems to many computer science undergraduates. Low self-efficacy in computer programming has been identified as one of the factors responsible for the observed problems which students encounter while learning programming. Little attention has been paid by researchers towards isolating factors that are likely to influence programming self-efficacy and achievement. This study, therefore examined the predictive value of intrinsic (gender, computer experience, mathematics background, computer ownership, locus of control, background in C++ and number of programming courses) and extrinsic (institution type) factors in undergraduates' self-efficacy and achievement in Java computer programming (SEiJCP and AiJCP) in south-western Nigeria.

The study adopted a correlational design. Purposive sampling technique was used to select 254 computer science undergraduates from four universities (three federal and one state). Only universities offering Java and C++ programming languages in their curriculum participated in the study. Five research instruments namely: Computer Experience Scale (r = 0.84), Java Programming Self-Efficacy Scale (r = 0.96), Java Programming Achievement Test (r = 0.70), Levenson Locus of Control Scale (r = 0.88) and Computer Background Questionnaire with four subscales: C++ background (r=0.87), computer ownership (r=0.90), mathematics background (r=0.84) and computer experience (r=0.79) were used to collect data. Data were analysed using descriptive statistics, t-test and Multilevel Analysis Procedures (MLAP). For MLAP, Null and Linear Growth Models were examined.

Null model showed that 91.0% of the variations in SEiJCP were due to institutional level differences. Students in the state university obtained higher scores in SEiJCP ($\bar{x}$=173.97; SD =26.39) than their colleagues in the Federal ($\bar{x}$=128.05; SD =44.57). The mean difference in SEiJCP scores between students in Federal and state universities was statistically significant (t = 7.57, df=252, p<0.05). The Null model also showed that 82.0% of the variations in AiJCP were due to institutional level differences. However, there was no significant difference in the mean score in AiJCP between students in the state university ($\bar{x}$=22.92; SD =11.78) and their colleagues in Federal universities ($\bar{x}$=20.54; SD =18.72). The Linear Growth Model (LGM) showed that only the number of programming courses significantly predicted SEiJCP (β=1.15), while gender, computer ownership, mathematics background, C++ background, computer experience and locus of control did not contribute significantly to the prediction. LGM showed that none of the intrinsic factors contributed to the prediction in AiJCP.

The number of programming courses significantly predicted self-efficacy in Java computer programming while institutional type significantly predicted both self-efficacy and achievement in it among computer science undergraduates in south-western Nigeria. Computer science departments in Federal and state universities should increase the number of programming courses in their curriculum. The Federal universities should also organise tutorial classes in all programming courses in order to improve self-efficacy in them.

Keywords: Java programming language, Computer science undergraduates, Self-efficacy, Institutional level differences, South-Western Nigeria

Word Count: 487

CERTIFICATION

This is to certify that this work was carried out by Owolabi, Josiah in the Institute of Education, University of Ibadan, Nigeria.

....

Supervisor Date

Supervisor
Benson Adesina Adegoke PhD
Senior Research Fellow
Institute of Education
University of Ibadan

ACKNOWLEDGEMENTS

Writing this dissertation would not be possible without the support of many people. Great appreciation is expressed to all those who offered me assistance and words of encouragement in the course of this work.

Firstly, I want to thank my supervisor, Dr. Benson Adesina Adegoke for his constant support, thoughtful suggestions and guidance. He possesses those qualities that students highly value in a mentor: a keen intellect, a taste for excellence, a friendly disposition and a genuine interest in seeing students succeed. He exhibited enthusiasm for my work, made available some relevant materials on Multilevel Analysis and Lisrel 8.80 software, encouraged me, critically read my drafts and gave me many helpful and insightful suggestions. Thanks and God bless you sir. I would like to thank Dr. J.G. Adewale who showed special interest in the work from inception, took off time to study the proposal and made several positive contributions to the work. The efforts and contributions of Dr. Isiugo-Abanihe is highly appreciated. Your invaluable suggestions have gone a long way to make the work what it is today.

My sincere gratitude goes to Dr. Eugenia.A. Okwilagwe and Dr. Monica Odinko who assessed the work at the initial stage. I can never forget the useful suggestions of Dr. J.O. Adeleke after going through part of the work and his encouragement and advice at various stages. I am indebted to Drs. Ikinat Junaid and J. O Abijo for the words of encouragements at the times I needed them.

My sincere thanks also go to the following: Professor T.W Yoloye for his concern and prayers, Professor E.Adenike Emeke for her motherly role in my academic and professional endeavours. I also thank very much all the other research fellows at the Institute of Education, University of Ibadan: Professor Promise Okpala, Professor (Venerable) Jonathan E. Ibeagha, Dr. Abiodun Adegbile, Dr. Georgina Obaitan, Dr. Faloyogun Falaye, Dr. Adams Onuka, Dr. Felix Ibode, Dr. Modupe Osokoya, Dr. C.V Abe and Dr. A.O Otunla. I have learned valuable lessons from each of them and they are worthy role models for upcoming scholars and evaluators. I appreciate the contributions and encouragements of Pastor (Dr.) D.O Ashamu, my senior colleague at the Institute.

I would like to gratefully acknowledge the support and assistance of my colleagues at the Institute, for encouraging me during the tough process and also attending my seminar presentations. Mrs Olayemi.O. Oshin, Mrs Olasunbo Olatunji, Mr Olonade P.O and a host of others too numerous to mention were all wonderful. Special thanks to Auntie Ronke and Sunday for their support. Thanks also to all my colleagues at the Mathematics/Statistics department of Federal College of Education (Technical), Akoka, Lagos, for their encouragement, understanding and support.

A special gratefulness is accorded to Professors Okunuga S.O and Olaleru J.O, both of Mathematics Department, University of Lagos. Dr. A.B.C Roberts and other lecturers in the Computer Science Departments of the University of Ibadan, Ladoke Akintola University (LAUTECH), Ogbomoso, Adekunle Ajasin University, Akungba-Akoko (AAUA), Ondo state, University of Lagos, Federal University of Technology, Akure and Obafemi Awolowo University, Ile-Ife, Osun state without whose supports data collection would not have been possible. My sincere appreciation also goes to BabajideOlusi , my wonderful research assistant for a job well done. My appreciation to all the participants of this research in all the five universities used for the study. God bless you all.

Mr. Peter Olanipekun, Miss Jemima Osanyinro, Mr. Tolulope Ilesanmi and Mr. Emmanuel Adeyemi are highly appreciated for their invaluable contributions to the work.
I also extend my gratitude to Pastors Victor Sopeju, Ukwa, Andrew Umoru, Ademola Farinmade, Tunde Ajao for the Pastoral concern and prayers. My prayer partners Brother Kudus Olaitan, Emmanuel, Peter etc. were wonderful. God bless you all.

My late parents (Pa Joseph Owolabi and Mama Mary Owolabi) are highly appreciated for the legacy of education. My profound gratitude goes to my brother and his wife; Engineer and Mrs Olufemi Owolabi, my sisters; Mojirade Alli and Rebecca Abba for their concern, prayers and motivation. You effectively gave the support my late parents would have given me if they were alive. The blessings of the Lord would never cease in your homes in Jesus Name.

This work would not have been completed without the faithful and continuing support of my darling wife, Janet Owolabi and our wonderful children, Jemima, Joshua and John Owolabi.

They have all endured a lot of inconveniences and denial of my much needed company during the course of this study. I appreciate you all.

Above all, my utmost thanks to God Almighty for the completion of this work, for every grace and favour received. To him alone are all glory and adoration even now and forever more. AMEN.

DEDICATION

This work is dedicated to God Almighty

TABLE OF CONTENTS Pages

CHAPTER FOUR: RESULTS AND DISCUSSION

CHAPTER FIVE: SUMMARY OF FINDINGS, IMPLICATIONS, RECOMMENDATIONS AND CONCLUSION

LIST OF TABLES AND FIGURES

TABLES PAGES

CHAPTER ONE

INTRODUCTION

1.1 Background to the Problem

Computer programming skill is a major aspect of computer science which is needed not only by the computer professionals but also by the non-computer professionals. Though the non-computer professionals are not required to engage in rigorous programming tasks, they may however need the programming skills at their work especially in the present information society.For computer professionals, acquisition of programming skills is inevitable. A computer is quite useless unless it is running a program. According to Jenkins (2004), programming lies at the very heart of computer.Pioro (2004) opines that programming courses are not just about programming perse, but that they provide a forum for teaching precise and logical thought processes. Moreover, they constitute necessary background for computer science students by introducing basic concepts and techniques to be used and to be built upon in more advanced computer science courses. An understanding of how the programs are written is a key part of the development of any computer science student. It is therefore not surprising that computer undergraduates are required to take and pass some programming courses during the course of their traning.

Computer programming is the craft of writing useful, maintainable and extensible instructions which can be interpreted by a computer system to perform a meaningful task. Precisely, Jenkins (2001) defined programming as “the process of taking a problem specification written in plain language, understanding it, devising a solution, and then converting the solution into a correct computer program (usually expressed in some special-purpose programming language)”. To program using the computer, one must learn how to give instructions to it. One must also learn the language understood by the computer. The instructions you give to the computer must be according to some specified rules. The words which make up the instructions as well as the rules which the instruction must obey form the computer language. In giving instructions to the computer, it must be done in any of the computer programming languages. Several computer programming languages had been developed and are still being developed.

The first set of computer languages developed were the machine languages. Machine languages were machine dependent (i.e they could be used on only one type of computer).It was however discovered that the machine language was slow for the programmer to write, time and effort wasting and proned to error. Consequently, the assembly language consisting of English – like abbreviations used to represent basic operations on the computer instead of the use of strings of numbers (as in machine language) was developed. Translator programs called assemblers were also developed to convert assembly language programs to machine language to enable the programs to be understood and processed by the computer.

A further developmental stage of computing and programming gave birth to the high level languages. The high level languages are more understandable by human beings. Just as in the case of assembly language, compilers are required to convert highlevel languages to machine form and vice versa. The earliest high level language developers used the procedural programming approach. The procedural programming separates the data of the program from the operations that manipulate the data. Examples of programming languages that used this approach are GW Basic, Q Basic, Fortran, Ada, Cobol, Algol, Logo, Pascal, C etc. A more recent programming paradigm is the object – oriented (OO) programming paradigm. The advantage of the object – oriented programming paradigm is that the data and the operations that manipulate the data (i.e the code) are both encapsulated in the object. For instance when an object is transmitted acrosss a network, the entire object (including the data and the operations) go with it.

There has been a number of publications that looked into the issues relating to the object – oriented (OO) programming in contrast to the procedural paradigm. For instance Robins, Rountree and Rountree (2003) submits that the object oriented approach is (i) Natural; (ii) easy to use and (iii) more powerful. Examples of the OO programming languages are C++, C#, Java, Visual Basic. Net etc. Of these object oriented programming approaches, C++ and / or Java languages are taught presently in most public universities in south – west, Nigeria. Java was chosen for this study because it is an offshoot of the C++. Besides, it is more relevant in the industries today and works on the web browsers.

It is ironical that the introduction of computer science degree in Nigerian Universities is yet to make any significant impact on the programming skills acquired by computer graduates. This is obvious from the level of participation in software development by these graduates. According to Jegede (2009a), most of the software packages presently in use in Nigerian industries, schools and financial institutions are either foreign (developed outside Nigeria) or locally adapted. Since computers are useless without software (which are basically programs) and there can be no programs without programmers, the need for competent and effective programmers cannot be over emphasized.

One reason for the dearth of programmers inspite of the many computer graduates produced from our institutions could be a result of the difficulties encountered by students in the programming class and the consequent high failure and drop out rate. Guzdial and Soloway (2002) estimated that 15 – 30 percent of students enrolled in an introductory course either drop out of the course or fail it. Reports by Mckinney and Denton (2004) even cited higher drop out and failure rates. The high drop out and failure rates are now a concern to all computer educators and the Information Technology (IT) professionals. This is because, it has the potential of causing shortage of professional software developers. This unfortunate development therefore necessitates the need to investigate some factors peculiar to individual students and their institutions which are capable of influencing students' achievement and self efficacy in programming.Programming is not a venture that one undertakes and concludes in a hurry. To write a program, one must: (i) understand the problem at hand. To solve a problem or process a data using the computer, the programmer must understand what he wants to do, (ii) design the computer program. The programmer must either receive or write a detailed specification of the solution to the problem and then represent the steps, using flowchart; (iii) code using a computer language; (iv) test and debug and (v) document the program.

Programmingrequires great efforts and perseverance. How people behave can often be better predicted by their beliefs about their capabilities than by what they are actually capable of accomplishing. This is simply because these beliefs help in determining what individuals do with the knowledge and skills they have (Bandura, 1986). Therefore, it is important to understanding what affects students' willingness to engage in programming tasks. According to Bandura (1986), people's judgment of their capabilities to organise and execute courses of action required to attain designated types of performances strongly influences the choices people make, the effort they expend, and how long they persevere in the face of challenges.

People's judgement of their capabilities to organise and execute courses of action required to attain designated types of performance is what is referred to in the literature as self efficacy. Self efficacy is therefore based on self perceptions regarding particular behaviours. It can be defined as the belief a person has about his ability to perform a particular task or behaviour (Bandura, 1977b). Self efficacy is therefore domain or task specific. Self efficacy is an important psychological construct which requires attention in research as it influences (i) the choice of activities that an individual takes part in; (ii) the amount of effort they will expend in performing a task and (iii) how long they will persevere in the face of stressful situations in completing that task (Bandura, 1977b).

Bandura (1994) explains that self efficacy beliefs determine how people think, feel, motivate themselves and even how they behave. He further explains that people with a strong sense of self efficacy view challenging problems as tasks to be mastered, develop deeper interest in the activities in which they participate, form a stronger sense of commitment to their interest and activities, and recover quickly from setbacks and disappointments. People with a weak sense of self efficacy avoid challenging tasks, believe that difficult tasks and situations are beyond their capabilities, focus on personal failures and negative outcomes and quickly lose confidence in personal abilities. Schwarzer (2004) submits that high self efficacy can enhance motivation. He further submits that people with high self efficacy set themselves higher goals, invest more efforts, show more resilience and persist longer than those with low self efficacy. Research on self efficacy theory has powerful effects which are embedded in social cognitive theory, positing that confidence in completing behaviours of interest will lead to achievement of those behaviours (Bandura, 1986). According to social cognitive theory, self efficacy influences an individual's interests, goals, and ultimately performance.

According to Hassan (2003), of the various factors that affects individual's willingness and ability to interact with computers examined in past research, computer self efficacy (CSE) has been identified as a key determinant of computer related ability (including programming) and use of computer. It is derived from the general concepts of self-efficacy and it refers to an individual's perceptions about his or her ability to use computer to perform a computing task (programming inclusive) successfully (Bandura, 1996&Compeau & Higgins, 1995).

Therefore, self-efficacy beliefs can affect how or whether a particular task or course of action will be attempted, and it may be a factor in whether people choose to get involved or not, in computer related activities (programming inclusive). Since it affects whether people attempt and /or persevere with the usage and study of computers, self efficacy is very important in computer science education research. Moreover, Marakas, Yi & Johnson (1998) suggest that computer self efficacy affect not only a person's perception of his or her ability to perform a computing task but also his or her intention towards future use of computer.

Research findings show that higher levels of perceived self efficacy correlate to greater motivational efforts and perseverance (Cassidy & Eachus, 2002). According to Bandura (1996), Compeau & Higgins (1995), computer self efficacy has proven to be a factor in understanding the frequency and success with which individuals use computers. Self efficacy theory, according to Askar & Davenport (2009) has emerged as an important means of understanding and predicting a person's performance. Jegede (2007) also opines that higher levels of computer self-efficacy correspond to greater achievement of computer competence. Social cognitive theory posits that a strong sense of self-efficacy leads individuals to undertake challenging tasks, expend greater effort in accomplishing a given task andpersist longer in the face of adversaries (Bandura, 2006a).

Given the research evidence on the influence of self-efficacy outlined earlier, it is reasonable to think that high self-efficacy in computer programming might play an important role in learning and writing programs and consequently producing competent and effective programmers in our nation. According to Wahab, Farhan,Norwawi, Hibadullah &Zaiyadi, (2010), learning to program is not an easy task to many students. The failure rates among students in higher institutions taking programming courses were also noticed to be as high as 30% to 40% (Shukur, Alias, Hanawi & Arshad, 2003 and Norwawi, Hibadullah & Osman, 2005). The general assumption that bright students can be successful in computer programming has been found not to be valid in some classrooms. According to Byrne and Lyons (2001), students who are proficient in other subjects sometimes perform poorly in programming. Hence the need to focus research studies on variables related to achievement in programming.

Although there is a correlation between perception and ability, the perception of the ability to successfully execute a task is independent of the actual ability.There is therefore the need to research into the two variables. Self-efficacy and achievement cannot be treated in isolation. They are influenced by several other variables as evidenced in literature (Cassidy & Eachus, 2002; Beas &Salanova, 2006). Some of these variables which will form part of this study are: Computer experience, locus of control, gender, mathematics background, computer ownership, C++ background and number of programming courses before entering the Java programming class.

There are several studies that had examined factors affecting General Computer Self Efficacy beliefs. Some of the studies found significant and positive relationship between computer experience and Computer Self Efficacy (Potosky, 2002; Hsiao, Lin & Tu, 2010). Karsten & Roth (1998) in their studies however found that computer experience had no significant impact on Computer Self Efficacy (CSE) beliefs.A few others others also examined factors relating to Java programming achievement and self efficacy (Askar & Davenport, 2009; Jegede, 2009a; Jegede, 2009b). Among the various variables examined as antecedents to general computer self efficacy, computer experience appeared prominent (Bandura, 1986). Social cognitive theory contends that prior experience represents the most accurate and reliable source of self efficacy information towards similar tasks. However the empirical result of the relationship between computer experience and computer self efficacy are inconclusive.

The contradictions in the findings from the previous studies might be due to lack of attention to the multi-leveled and multifaceted nature of computer self efficacy. Computer self efficacy evolved to investigate both general and specific computer self efficacy and the relationship between them (Agarwal, Sambamurthy & Stair, 2000). General computer self efficacy is defined as an individual's judgement of efficacy across multiple computer application domains while specific computer self efficacy is defined as perceptions of ability to perform specific computer-related tasks in the domain of general computing. The focus of this study is not just on general computing but on a specific computer task (Computer Programming).

Studies on the relationship between computer experience and computer programming self efficacy and achievement are very rare. Askar & Davenport (2009) in a study of factors related to Java programming self efficacy among engineering students in Turkey found out that the number of years of computer experience had a significant linear contribution to Java programming self efficacy scores. However, Jegede (2009a), in a similar study among engineering students in a University in south- west, Nigeria found that the number of years of experience in programming did not significantly predict Java programming self efficacy scores. The empirical inconsistency reported in the two studies that borders on the relationship between computer experience and specific computer task (Java programming self efficacy may be attributed to the fact that computer experiences of an individual is bound to vary across different computer tasks. While Askar & Davenport (2009) used the general computer experience as a single construct reflecting the number of years of computer use; Jegede (2009a) was specific-he used the number of years of programming experience.

Previous studies suggest that computer experience represents a multi-dimensional construct that comprises various experiences with computer applications and software tools and that specific computer experiences offer more accurate and reliable predictors of self efficacy than the general single-dimensional computer experience construct (Bozionelos 2001; Hoxmeier, Nie & Purvis, (2000). Only few researchers examined the impact of different types of computer experiences on both General and specific Computer Self Efficacy (CSE) beliefs. Hassan (2003) showed that certain computer experiences has varying levels of impact on a person's perceived computer self efficacy. Hassan's research showed that experiences with computer programming and graphic applications had strong and significant effects on computer self efficacy beliefs while spreadsheet and database applications demonstrated weak effects.

Busch (1995) specifically investigated the influence of experience with computer programming, computer games, word processing and spreadsheet applications on task specific self efficacy beliefs towards Lotus 1-2-3 and word perfect applications. The results showed that experience with word processing applications was the most influential predictor of self efficacy beliefs with regard to word perfect while computer programming had the strongest effects on self-efficacy beliefs towards Lotus 1-2-3. Jegede (2009a) narrowing down his investigation to the relationship between number of years of programming experience and Java programming self efficacy among engineering students in a South-Western Nigerian University found that, the number of years of experience in programming did not significantly predict Java programming self efficacy scores.

Several studies have been conducted to find out reasons for poor levels of achievement in programming. One reoccurring factor in most of the research reports that seem to influence programming achievement is prior computer experience. In particular, prior programming experience has been found to affect computer programming achievement (Koohang & Byrd, 1987). Taylor & Mounfield (1989) also found that prior experience in programming provides a significant predictor of how students perform in the programming courses. They found that prior exposure whether at the high school or college level is an important factor in students' performance in computer programming.

Identifying the computer experiences that have stronger impacts on self efficacy and achievement scores in recent and current programming styles among computer majors in our universities will provide reasons why students perform poorly in programming examinations and lack confidence in their ability to program using the recent approaches. It will also provide insight into designing effective training programs to enhance their programming self efficacy and achievement and have more of them in software development industry after graduation. This research among other thingsconcentrated on examining the relationship between specific computer experiences (Word Processing, Spreadsheet, Database, Operating Systems, Graphic, Games, Telecommunications, Internet, Programming experiences) and self efficacy and achievement in Java programming.

Locus of control (LOC) has also been identified as a factor that could influence self efficacy and achievement. It consists of two dimensions of causes: Internal and External. Those with an Internal LOC generally expect that their actions will produce predictable outcomes. Those with an external LOC generally expect that outcomes are due to external variables such as fate, luck or powerful others. The initial scale of measurement was that of Rotter (1966). This was consistent with the conceptualization that only the Internal LOC and External LOC exist. However, a series of inconsistent findings in the 1960s and 1970s led to calls for revision of this initial scale (Joe, 1971; Lefcourt, 1972).

Levenson (1974) proposed that the inconsistencies in findings across research studies were among other reasons for the revision of the scale. There are actually two types of external LOCs: (i) those who believe that the world is ordered and powerful others are in control; and (ii) those who believe that the world is unordered and events are due to non-human forces (such as chance or fate). The scale developed and validated by Levenson (1974) was designed to address these concerns and is now a common alternative to the standard Rotter scale. The Levenson scale uses a Likert scale thus allowing the dimensions to be statistically independent. The external dimension is now categorized as either a belief that control is in the hands of human force, (i.e powerful others) or non-human forces (i.e chance). For those individuals who believe in powerful others, outcomes are predictable and the potential for control exists. For those who believe in chance, outcomes are unpredictable and control is not possible. It is now generally accepted in the psychological literature that LOC is a multidimensional construct (Skinner, 1996) and that Internal, Powerful others and chance control are theoretically independent constructs.

According to Araromi (2010), those with a high internal LOC believe that events results primarily from their own behaviour and actions. Consequently, they are likely to assume that their efforts will be successful and are more likely to be active in seeking information and knowledge concerning their situation. This is unlike those with a high external LOC who believe that powerful others, fate or chance primarily determines events. Emeke and Yoloye (2000) in a study conducted to compare the adjustment of foreign students across their Locus of control groupings, the continent of their origin and gender found that internally controlled students were better adjusted than the externally controlled ones. A number of previous studies identified significant relationships between LOC and academic achievement. Emeke, Adeoye and Torubeli (2006) in a study found a relationship between locus of control and achievement to be high, positive and significant ($r = 0.787$, $p < 0.05$). Fakeye (2011) conducted a study to investigate the relationship between LOC and achievement in English Language. The finding of the study showed a difference (which is not significant) between the achievements of internal and Exteranl LOC. Those with Internal LOC in the sample performed better than their counterparts with external LOC. Stubbs (2001) in a study conducted concluded that internals tend to show superior achievement compared to their external counterparts.

A number of studies have examined relationship between LOC and programming skills. Bishop-Clark (1995) identified LOC as a factor that may help in explaining the variability in programming skills acquisition. Chin & Zecker (1985) observed that internals were more likely to succeed at programming than externals. Jegede (2009b) found that programming achievement has no significant relationship with the faith students have in their own lives (internality), the belief in the irresistible power of others on their lives (powerful others) and the trust they place on chance in determining their course in life (chance).

Literatures on studies investigating the relationship between LOC and programming self efficacy and achievement seems to be rare, in view of this, there is the need to examine the relationship between LOC and programming self efficacy and achievement among computer majors in Nigerian universities. This may assist in better direction of efforts towards advancing appropriate logistics that will enhance high programming self efficacy and achievement among computer majors and consequently bring about better development in the IT industry. It is on this note that this study will also examine the relationship between computer undergraduates' LOC and their self efficacy and achievement in Java programming language.

In research, relationship between gender and computer self efficacy has been of regular interest, possibly because the computer was seen as a skill area for the male folks. So far, findings on gender influence on computer self efficacy are mixed. Some studies showed that males evidenced higher self efficacy than females (Harrison& Rainer, 1992);Jegede (2007) however found no gender differences in computer self efficacy. Busch (1995) observed gender differences in perceived self efficacy regarding completion of complex tasks in both word processing and spreadsheet software (with males having higher CSE scores). In the same study, no gender differences were found in self efficacy regarding simple computer tasks.

The influence of gender on programming achievement is also inconclusive. Linn (1985), in a study organized among middle school programming classes, found that girls and boys have similar levels of programming achievement once they enroll in classes, but that girls are less likely to enroll. Yukselturk & Bulnut (2007), in a study observed that gender was among the variables excluded from the equation of predicting success in an online computer programming course because it did not have a significant contribution to variance in success ($p > 0.05$). Askar

& Davenport (2009) in a study of factors related to self efficacy for Java programming among engineering students in Turkey found that males evidenced higher programming self efficacy. Gender studies on computer programming self-efficacy and achievement seems to be rare especially in Nigeria. This study shall therefore investigate the influence of gender on Java programming achievement and self efficacy among computer majors in South-Western Nigerian universities.

Mathematics achievement has also been identified in literature as a factor that affects computer programming. A possible reason for this could be because mathematics problem solving and programming require similar skills and ability to succeed. According to Harkins (2008), problem-solving strategies employed in a traditional college mathematics course are essentially the same in a first course in computer programming. The primary difference is that in computer programming the problem's solution is implemented on a machine using a computer language to direct the solution. It could therefore be said that the logical reasoning and critical thinking skills which are so vital to success in mathematics are likewise crucial to success in computer programming.Moreover there had been a concensus that mathematics ability predicts performance in programming. Wilson (2002), Byrne and Lyons (2001) inseparate studies of factors contributing to success in computer programming achievement found that mathematicsachievement is one of the key factors that could predict achievement in programming.Mathematics ability, measured as achievement is however different from mathematics background. The number of mathematics courses taken by the respondents before the study was used as the mathematics background. There seems to be dearth of studies on the influence of mathematics background on achievement and self efficacy in programming.   This study will focus on how mathematics background predicts both achievement and self efficacy in programming.

The relationship between computer ownership and computer self efficacy has never been found to be consistent in previous findings. Tokzadeh and Koufterous (1994)as well as Houle (1996) in their separate studies found that owning a computer is significantly correlated with computer self efficacy. Busch (1995) found that students who have access to their own computer cooperated more in front of the computer than any other group. However, Cassidy & Eachus (2002) found that owning a computer was not a significant predictor of computer self efficacy.

Computer ownership generally has been found to influence usage. Only few of the students who do not possess computer technologies attempt to access and use them through other means. A report by Rural Education Action Project (2010) on ownership, access and use of computers, Information Technologies and other e-technologies in Beijing schools indicated that 36% of the respondents reported that they access the internet through other family members and only 1% access the web at internet cafes. The implication of this is that: students who possess personal computers and other technological devices use them very often; students in families that own computers and other technologies use computers often; and students without ownership (either at personal or family level) may not use them at all. Since computer ownership influences usage generally. For a computer student in the programming class who is exposed to programming experiences and owns a computer; it is reasonable to think that he will engage in programming more than the course mate without a computer; since programming art is appreciated more when carried out on the computer. This study will therefore examine the pattern of influence of computer ownership on Nigeria undergraduates' achievement and self efficacy in Java programming languages.

Java programming language happened to be the offshoot of the C++ programming language. Some Universites teach it to the students who are later taken through Java programming. Some universities do not. Research has however shown that exposure to certain programming languages influenced performance in Java programming. Hoskey and Maurino (2010) conducted a study among all two hundred students who took Java programming language between 2005 and 2009 fall semester. Even though these students all took Java programming language; prior to their Java classes, some were exposed to C++ programming language and some others to Visual Basic. The finding showed that those previously exposed to C++ did better than those exposed to Visual Basic. Since Java programming language is the offshoot of C++, it forms the prerequisite required to learn Java programming language. It is therefore reasonable to conclude that those taught C++ before entering Java programming class would perform better and have higher self efficacy in Java programming language. It is on this note that this study sought to examine the extent of the influence of C++ background on achievement and self efficacy of computer undergraduates in Java programming.

The curriculum of universities is flexible especially because of the way it is implemented in the different universities. Specifically, the number of courses computer undergraduates are exposed to before entering the Java programming class vary from one university to another. Schonberg and Drivar (2008) opined that Java should not be introduced as an introductory programming course. According to him, Java hinders the understanding of code performance; the design methodologies of Java lead to a proliferation of objects, heavy use of dynamic storage and data structures are pointer heavy and this considered wasteful. Also studies on the influence of number of programming courses taken before entering Java programming class seems to be very rare in literature. This study will also investigate the extent of prediction of Java programming achievement and self efficacy by number of programming courses taken before the Java programming class.

Wiedenbeck (2005) observed that perceived self efficacy in programming affected performance in programming courses. There seems to be no research report yet on factors predicting self efficacy and achievement in computer programming in Nigeria. No doubt, computer undergradutes'self efficacy are functions of many factors ranging from intrinsic to extrinsic factors. The intrinsic factors are those in which the student himself or herself has a significant input or part of the real nature of the student such as his or her gender, computer experience, locus of control, mathematics background, computer ownershipand number of programming courses before entering Java class; while the extrinsic factors are those in which the student has little or no significant input, or not part of real nature of the student such as the type of institution which he or she attends.

Since it is not possible to investigate the self-efficacy of all programming languages in a single study, specifically this study focussedonJava programming language. The choice of Java is necessitated because it is one of the most recent programming languages taught in our public universities that are platform independent (it runs on more than one platform without needing to be recompiled). A platform is a type of computer operating system like windows, Mac OS, Linux. It can also run on a web browser. Besides, it is still relevant in the industry.

As it is in many educational researches, it became necessary in this work to investigate the relationship between individual students and their institutions and vice versa. It is a general belief that individuals interact with the social contexts to which they belong. The implication of this is that individual persons are influenced by the social groups they belong which are in turn influenced by the individuals who make up that group. The individuals and their groups are conceptualised as hierarchical system. The individuals and their groups are defined at different levels of the hierarchical system. According to Kreft and Deleeuw (1998), once you know that hierarchies exist, you will see them everywhere. Whether by design or nature, most educational researches often use data sets that are referred to as "nested" or hierarchically nested. This is because observations at one level are nested within observations at another level.

In this work, the computer undergraduates that served as the subjects are nested within institutions. The students' variables formed the level 1 variables (micro level) while their institutions constituted the level 2 variable (macro level). This type of data are referred to as a multilevel data. In this study the researcher used techniques that takes into account the nesting. The results of the analysis that do not take into account the multilevel nature of the data may (or perhaps will) be inaccurate (Nezlek 2001, 2007).

In this study where students are nested within their institutions, aggregation of students'characteristics over their institutions would facilitate an institution analysis. There is no doubt that in the process, all individual information is lost. Most times, within group variation accounting for most of the total variation in the outcome is lost. The loss of individual information therefore can have an adverse effect on the analysis and also lead to distortion of relationships between variables. Another option is to disaggregate the data by assigning institutional data to individual students. By this, the assumption of independent observation would no longer hold. In hierarchical data, individuals in the same group are also likely to be more similar than individuals in the different groups. Therefore the variations in outcome may be due to difference between groups and to individual difference within a group.

1.2. Statement of the Problem

Research has shown that learning computer programming is one of the problems many computer undergraduates face in the course of their studies in computer science. Literature has also shown that the failure rate among computer undergraduates in computer programming courses is high. Moreover, many who manage to pass computer programming courses demonstrate inadequate competence in programming. Studies have tried to identify reasons for the poor performance and lack of adequate competence in computer programming among computer undergraduates.

Among various factors identified, computer self efficacy has been identified as a key determinant of performance and competence in computer programming. However, it seems that from the literature, researchers have shown little or no interest in identifying factors that are likely to influence undergraduates'computer programming self efficacy and achievement.Moreover, studies that have used multilevel analysis to predict undergraduates' computer programming self efficacy and achievement appear to be very few. It is on the basis of this that the researcher sought to examine the extent to which intrinsic factors (gender, computer experience, computer ownership, mathematics background, Locus of control, background in C++, and numberof programming courses taken before entering Java programming class) and extrinsic factor (type of institutions) can predict undergraduates' self efficacy as well as achievement in programming. It is on the basis of this that the researcher also sought to use multilevel analysis of intrinsic and extrinsic factors to predict undergraduates'self efficacy and achievement in computer programming in south west, Nigeria.

1.3. Research Questions and Hypothesis

Research Questions

The study sought answers to the following research questions.

1. What type of relationship exist among intrinsic factors (gender, computer experience, computer ownership, mathematics background, Locus of control, background in C++, and numberof programming courses taken before entering Java programming class), extrinsic factor (type of institution)and Java programming self efficacy?
2. Whattypes of relationship exist among the investigated intrinsic factors, extrinsic factor (type of institution)and Java programming achievement?
3. How much of the total variance in Java programming self efficacy of computer undergraduates is accounted for by institution-level and student-level differences?
4. How much of the total variance in Java programming achievement of computer undergraduates is accounted for by institution-level and student-level differences?

5. How much of the student-level variance in Java programming self efficacy of computer undergraduates is associated with gender,computer experience, computerownership, mathematics background, background in C++, Locus of control and number of programming courses taken before entering Java class.
6. How much of the student-level variance in Java programming achievement of computer undergraduates is associated with gender, ownership of computer, mathematics background, background in C++, computer experience, Locus of controland number of programming courses taken before entering Java class.

Hypotheses

1. There is no significant difference, in the mean score of self-efficacy in Java Programing, between undergraduates in Federal and State Universities.
2. There is no significant difference, in the mean score of achievement in Java Programming, between undergraduates in Federal and State Universities.

1.4 Scope of the Study

Java programming languages formed the focus of this study. Computer science students who had been exposed to Java programming language were sampled for the study. Only government owned (federal and state) Universities in the southwestern part of Nigeria that have Java programming language in the computer science curriculum were used for the study. The influence of computer undergraduates'intrinsic and extrinsic factors on computer self efficacy and achievement in computer programming were examined.

1.5 Significance of the Study

The findings from the study would afford the different stakeholders (e.g computer lecturers, the employers in the Information Technology Industry especially those who are into software development, directorate of academic programmes in the universities, parents, policy makers and the government) the opportunity to know what factors could influence computer students' self efficacy and achievement in Java programming language. Based on the findings, recommendations that can help increase students'self efficacy and achievement in computer programming generally and Java programming in particular were suggested. Increasing self efficacy and achievement in Java programming will (i) help the students to successfully go through every stress and difficulties involved in various programming tasks using Java language; (ii) help University graduates to be relevant in the industry; (iii) encourage them to take to programming and software development after graduation; (iv) consequently help in making Nigeria an IT capable country and a key player in the information society. The findings from this study will also help educational psychologists in guiding human behaviour in desirable ways towards effective learning and performance in programming as well as productivity in the world of work generally and the IT industry in particular. The multilevel analysis which is a more robust statistics compared to ordinary multiple regression used in this study will afford the researchers the opportunity to have insight into greater details of the pattern of influence of extrinsic and intrinsic independent variableson the dependent variables. Finally, the database from this study will provide further illumination into researches onself-efficacy and achievement in programming.

1.6. Operational Definition of Terms

a. **Intrinsic factors**: These are factors which the student himself or herself has a significant input in or part of the real nature of the student such as his or her gender, computer experience, locus of control, mathematics background, computer ownership and number of programming courses before entering Java class.

b. **Extrinsic factors**: These are factors which the student has little or no significant input, or not part of real nature of him or her such as the type of institution which he or she finds himself/herself.

c **Mathematics Background**: This includes the number of mathematics coursestaken by the computer undergraduates.

d. **JavaProgramming Achievement**: This is the array of scores of computer undergraduates in Java programming Achievement test. It is a continuous variable.

e. **Java Programming Self Efficacy**:Thisis the measure of the feeling about one's ability to performvariousJava programming tasks as measured by the Java Programming Self-efficacy Scale designed by Askar&Davenport (2009) and revalidated before it was used.

1.7 Acronyms and abbreviation

M.L.A – Multi - Level Analysis

I.T – Information Technology

L.O.C – Locus of Control

C++ – C Plus Plus

O.O.P – Object Oriented Programming

O.A.U – Obafemi Awolowo University

CHAPTER TWO

LITERATURE REVIEW

This chapter presents relevant literature that was reviewed.

Literature was discussed under the following headings:

2.1 Theoretical Background

2.2 Evolution and Development of Computer Programming.

2.3 Self Efficacy and Achievement in Computer Programming.

2.4 Intrinsic and Extrinsic Factors

2.5 Computer Experience, Self Efficacy and Achievement in Programming.

2.7 Gender, Self Efficacy and Achievement in programming

2.8 Computer Ownership, Self Efficacy and Achievement in Programming

2.9 Locus of Control, Self Efficacy and Achievement in Programming

2.9 Mathematics Background, Self Efficacy& Achievement in Programming

2.10 C++ Background, Self Efficacy and Achievement in Programming

2.11 Number of Programming, Self Efficacy and Achievement in Programming

2.12 Rationale for Multi Level Analysis

2.13 Appraisal of Literature Reviewed

2.14 Gaps of Literature Reviewed.

2.1 Theoretical Background

This study is underpinned by social cognitive theory. Self efficacy is grounded in the theoretical framework of social cognitive theory developed by Albert Bandura (1977b, 1997). This theory emphasizes the evolvement and exercise of human agency – that people can exercise some influence over what they do (Bandura, 2006b). This assumes that such human agency operates in a process called triadic reciprocal causation. Reciprocal causation is a multi – directional model suggesting that our agency results in future behaviour as a function of three interrelated forces. These forces are environmental influences, behaviour and internal personal factors such as cognitive, afective and biological processes.

The theory assumes that these (environmental influences, behaviour and internal personal factors such as cognitive, affective and biological processes) impacts its members, determines what we come to believe about ourselves and affects the choices we make and actions we take. According to the theory we are not products of our environment. We are not products of our Biology; instead we are products of the dynamic interplay between the external, the internal and our currentand past behaviours. According to Bandura (1986), dialistic doctrines that regard mind and body as separate entities do not provide much enlightenment on the nature of disembodied mental state or on how an immaterial mind and bodily events act on each other.

Central to this social cognitive theory is the concept of self efficacy. Bandura's aspirations about self efficacy were grand, as reflected in the title of his 1977 article "Self-Efficacy": Towards a unifying theory of behavioural change. In his seminar work, he defined self efficacy as "beliefs in one's capabilities to organise and execute the courses of action required to produce given attainments". Self efficacy beliefs were characterised as the major mediators for our behaviour and importantly, behavioural change.

Bandura (2006a) maintains that people are self organising, proactive, self regulating and self reflecting. According to Schunk and Meece (2006), self efficacy affects one's goals and behaviours and is influenced by one's actions and conditions in the enviroment. Also according to Bandura (2006a), efficacy beliefs determine how environmental opportunities and impediments are perceived. According to Pajares (1997), it affects choice of activites, how much effort is expended on an activity and how long people will persevere when confronting obstacles.

Based on social cognitive theory, a computer undergraduates Java programming self efficacy may be conceptualised as his or her belief in his own ability to write and execute Java programs successfully to solve a given problem. Based on this, Bandura (1997, 2006b) recommended the following for item construction: (i) because self efficacy is concerned with perceived capabilities, the items should contain verbs like "can" or "be able to" in order to make clear that the items ask for mastery expectations because of personal competence, (ii) the object in each statement should be "I" since the aim is to assess each computer undergraduate's subjective belief about his or her capability and (iii) each item should contain a barrier. The third point above is highlighted by Bandura (1997b) when he noted that "if there are no obstacles to surmount, the activity is easy to perform and everyone would have uniformly high perceived self efficacy for it". Askar and Davenport (2009) followed this recommendation in the design of the Java programming self efficacy scale used in this study.

Self efficacy helps individuals to succeed at tasks (Bandura, 1993). Even though knowledge and skills are required, Bandura (1993) reported that those requirements are not necessary to guarantee sucess. When two people have similar educational backgrounds and skill but are at different levels of self efficacy, one may succeed while the other will fail. The cause in this case will not be their educational background but their self efficacy.

Bandura (1994) stated that there are four main sources that influence a person's self efficacy: mastery experiences, vicarious experiences, social (Verbal) persuasion and somatic and emotional states in judging ones capabilities (Physiological arousal).The first and the most effective is the "mastery experiences" it could also be referred to as "successes at tasks". It increases self efficacy. Failure at taskson the other hand may inhibit self efficacy development. The mastery experience required to increase ones self efficacy is the one that takes time and effort to accomplish. The increase in self efficacy that comes through quick and easy tasks may not last. It may even lead to decrease. For instance, when a more challenging task arises, it may cause the person to become frustrated, stressed and consequently cause a decrease in self efficacy. Successes at programming tasks if achieved, according to social cognitive theory would increase the level of self-efficacy which would consequently increase the level of achievement. Therefore every strategy needed to ensure initial successes at programming tasks is inevitable.

The second source of strengthening self efficacy is vicarious experiences provided through observing colleagues performing similar tasks. Observing the success of others similar to oneself contributes positivelyto self efficacy. On the other hand observing the failure of others similar to oneself may decrease self efficacy.The third source of strengthening self – efficacy is through "social persuassion". Ones self efficacy is increased when one is told by others that he/she has what it takes to succeed. If one was also told that he does not have the skills for success, his self efficacy may decrease.The fourth and final source of self efficacy is through "sematic and emotional states in judging ones capabilities". According to Bandura (1997) Somatic indicators of personal efficacy are especially relevant in domains that involve physical accomplishments, health functioning and coping with stress". Relieving stress and enhancing physical status can increase self efficacy (Bandura, 1997).

Bandura (1977b) emphasized observational learning as a general process of acquiring information from another person, verbally and usually. This goes beyond imitation which involves mere literal duplication or behaviour. He believed that much of learning comes from observational learning and instruction rather than from overt trial and error behaviour. This becomes most relevant and related to perceived self efficacy, the idea which originated from Bandura himself. Obviously, a computer graduate that would judge himself competent to write programs (perceived self efficacy in programming) must have arrived at this stage through the four sources of self efficacy: (i) mastery experience; (ii) vicarious experiences; (by observing others doing it); (iii) verbal persuasion from role models and (iv) psychological traits.The social cognitive theory explores how people acquire and maintain certain behavioural pattern, while also providing the basis for intervention strategies (Bandura, 1977b).

The concept of behaviour can be viewed in many ways. Behavioural capability means that if a person is to perform behaviour, he must know what the behaviour is and have the skills to perform it (Glanz, Rimer & Lenis, 2002). The summary of this is that competence in programming will be made possible through consistencies and resilience in the art and practice of programming, which will be possible only when an individual concerned judges himself competent to program (self efficacy in programming). Successful programming activities also increase the individual's perceived self efficacy iin programming.

2.2 Evolution and Development of Computer Programming

Computer programming has gained so much popularity and attention in the workplace that even non-computer professionals make use of programming at work. According to Peyton – Jones, Blackwell and Burnett (2003), Rosson, Ballin and Nash (2004), Rothermel, Burnett, Dupuis and Sheretor (2001) and Wiedenbeck and Engebretson (2004) creating macros, spreadsheet formulas and dynamic web applications in work place require writing programs. Its history can be traced to the advent of computers. Most computer scientists who were actively involved in the evolution of computer are also important figure in the developmental processes of computer programming language. "Computer programming" is a phrase used to refer to the various processes of giving a set of instructions to the computer. Such processes include designing, writing, testing, debugging and maintaining the source code. Programmers write instructions in various programming languages; some can be easily understood by the computer while others require immediate translation steps.

Although hundreds of computer languages are in use today, they can be classified into three general types: machine languages, assembly languages, and high – level languages. Machine language is the natural language of a computer and it is defined by the computers hardware design. Machine languages are "machine - dependent". This implies that such languages can be used on only one type of computer. As the popularity of computer increased, machine-language program was discovered to be slow, time and effort wasting and prone to error. English – like abbreviations were used to represent the basic operations on the computer rather than using the strings of numbers. These abbreviations formed the foundation of assembly languages and translator programs called assemblers were used to convert assembly language programs to machine language at the speed of computer. Another developmental stage of computing and programming gave birth to the high – level languages. The use of high level languages enables accomplishment of tasks at greater speed. Compilers were active in converting or translating the high level programs into machine language at a very fast rate. Of course, programmers would prefer this kind of programming language that achieves much within a short time period. Specifically, programming languages suh as C, C++, Java, C# (pronounced as "C" Sharp), Python, etc are common high – level languages.

In this section, we shall take a cursory look at the historical development of high-level programming languages in general and then make special emphasis on C++ and Java.

2.2.1 History of Computer Programming Languages

The development of programming languages was not without a step – by – step or outlined procedure usually required for solving a problem that is an "algorithm". Knuth and Pardo (1976) reviewed that the earliest known written algorithms come from ancient Mesopotamia about 2000 B.C. At that time, the written descriptions contained only sequences of calculations on particular sets of data, not an abstract statement of the procedure. By the time of Greek civilization, several non-trivial abstract algorithms had been studied rather thoroughly (Knuth, 1976). They (Knuth and Pardo, 1976) further explained that during the ensuing centuries, mathematicians never did invent a good notation for dynamic processes, although of course, notations for (static) functional relations became highly developed. When a procedure involved non trivial sequences of decisions, the available methods for precise description remained informal and rather cumbersome. Some programs written for early computing devices, such as those for Babbage's calculating Engine were naturally presented in "machine language rather than in a true programming language". The most elatorate program developed by Babbage and Lady Lovelace for this machine was a routine for calculating Bernoulli numbers (Babbage, 1961). Also in 1914, Leonardo Tores Queredo used natural language to describe the steps of a short program for his hypothetical automation: and Helmut Schreyer gave an analogous description in 1939 for the machine he had helped Konrad Zuse to build (Randell, 1973). In addition, an example of MARK I program given in 1946 by Howard Aiken and Grace Hopper (cited in Randell, 1973) shows that its machine languages were considerably more complicated. Although all these early programs were in a machine language, it is interesting to note that Babbage had noticed already on July 9, 1836 that machines as well as people could produce programs as output.

Near the end of World War II, Allied bombs destroyed nearly all of the sophisticated relay computers that Konard Zuse had been building in Germany in 1936. Only his Z4 machine could be rescued, in what Zuse describes as a fantastic ("abenteuerlich") way; and he moved the Z4 to a little shed in a small Alpine Village Called Hinterstein. Other machines built by Zuse include: Z1 which he built in his parent living room in 1936; Z2 which he experimented with relays for the ALU; Z3 an all – relay technology that is, the first electronic programmable digit computer; Z4 which was envisioned as a commercial system.

Zuse had previously come to grips with the lack of formal notations for algorithms while working on his planned doctoral dissertation (Zuse, 1976). He had independently developed a three – address notation remarkably like that of Babbage, he however realised that this notation was limited to straight – line programs (starre plane) and he thus concluded his previous manuscript, with these remark: unstarre Rechenplane constitute the time discpline of higher combinational computing (Zuse, 1976). The result was an analysing comprehensive languages which he called the plankakul (program calculus). Before laying this project aside, Zuse had completed an extensive manuscript containing programs far more complex than anything ever written before.

However, much work on computer programming lnguages was not done until 1952 with short code for UNIVAC (Sammet, 1972). Before this time Charles Babbage had created a difference engine which could only be made to execute tasks by changing the gears that executed the calculations. Thus, the earliest form of computer language was physical motion. Eventually, physical motion was replaced by electrical signals when the US government built ENIAC in 1942. It followed many of the same principles of Babbage's engine and hence, could only be programmed by presetting switches and rewiring the entire system for each new program or calculation. This process proved to be very tedious (Ferguson, 2000).

In 1945, John Von Neumann developed two important concepts that directly affected the path of computer programming languages. The first was known as "shared – program technique" (http://www.softlord.com). This technique stated that the actual computer hardware should be simple and not need to be hand – wired for each program. Instead complex instructions should be used to control simple hardware, allowing it to program much faster (Ferguson, 2000). The second concept was also extremely important to the development of programming languages. Von Neumann called it "conditional control transfer" (http://www.softlord.com). This idea gave rise to the motion of subroutines or small blocks of code that could be jumped to in any order, instead of a single set of chronologically ordered steps for the computer to take. The second part of the idea stated that computer code should be able to branch based on logical statements as IF (expression) THEN, and loops such as with a FOR statement. "Conditional control transfer" gave rise to the idea of "Libraries", which are blocks of code that can be re-used over and over (Ferguson, 2000).

FORTRAN (FORmula TRANslating system) which was first of the major languages (Ferguson, 2000) or higher level language (Sammet, 1972) appeared in 1957 (Ferguson, 2000) was developed in 1954 (Verkeyn, 2005). It is pertinent however to note here that Ferguson's and Verkeyn's accounts of FORTRAN development (with respects to date) do not agree. This is one of the problems encountered when discussing history of programming languages due to the fact that "there are a number of phases (in developing a programming language) each of which is important in the overall development, but which is almost impossible in retrospect to pinpoint to an exact time" (Sammet, 1972) or date. FORTRAN was designed at IBM for scientific computing. According to (Ferguson, 2000), the components were very simple, and provided the programmers with low-level acess to the computers. Today, this language would be considered restrictive as it only included IF, DO and GOTO statements, but at the time, these commands were a big step forward. The basic types of data in use today got their start in FORTRAN, these included logical variables (TRUE or FALSE) and integer, real and double – precision numbers. Ferguson (2000) agrees with Sammet (1972) when the latter noticed that FORTRAN opened the door to practical usage of computers by large numbers of scientific and engineering personnel. Ferguson explains further that although FORTRAN was good at handling numbers, it was not so good at handling input and output, which matterd most to business computing.

Business computing became an isssue of special interest to the USA army and computer scientists. This informed the development of Common Business Oriented Language (COBOL) in 1959 (Ferguson, 2000; Verkeyn, 2005). It was designed from the ground up as the language for businessman and Colonel Grace Hopper was very instrumental to this development. It's only data types were numbers and strings of text. It also allowed for these to be grouped into arrays and records, so that data could be tracked and organised better (Ferguson, 2000).

In 1958, John Mc Carthy of MIT created the LISt Processing (or LISP) language. According to Verkeyn (2005) this language was designed for the manipulation of symbols and patterns (characters, words, etc) and also designed for Artificial Intelligence (AI) research. Ferguson notes that the original release of LISP had a unique syntax because it was designed for a specialised field. Programmers wrote code in Parse trees, which are usually a compiler – generated intermediary between higher syntax (such as in C or Java) and lower – level code.

Another obvious difference between this language (in original form) and other languages is that the basic and only type of data is the list; in the mid – 1960, LISP acquired other data types. The LISP syntax was known as "Cambridge Polish" as it was different from standard Boolean logic (Wexelblat, 1981): xVy – Cambridge polish, what was used to describe the LISP program; oR (x, y) – parenthesized prefix notation was what was used in the LISP program; x OR y – standard Boolean logic.

Another programming language that was developed in 1958 is Algol (ALGOrithmical Language). Edsger Dijksta, Nicolaus Wirth and Naur were active in the committee that developed Algol. It was designed to be the successor of FORTRAN (Venkeyn, 2005). (Ferguson, 2000) stated that Algol's major contribution is being the root of the tree that has led to such languages as Pascal, C, C++ and Java. It was also the first language with a formal grammar, known as Backus – Naar Form or BNF (McGraw – Hill Encyclopedia of Science and Technology, 1997). Though Algol implemented some novel concepts, such as recursive calling of functions, the next version of the language Algol 68 became bloated and difficult to use (www.byte.com). This led to the adoption of smaller and more compact languages, such as Pascal.

Pascal was developed in 1968 (Ferguson, 2000) or in 1971 (Verkeyn, 2005) by Niklaus Wirth. It's development was mainly out of necessity for a good teaching tool. In the beginning, the language designers had no hopes for it to enjoy widespread adoption. Instead, they concentrated on developing good tools for teaching such as a debugger and editing system and support for common early microprocessor machines were in use in teaching institutions (Ferguson, 2000). Pascal was named after Blaise Pascal who built the first mechanical calculation device which could only add and subtract. It was designed in a very orderly approach; it combined many of the best features of the languages in use at the time, COBOL, FORTRAN, and ALGOL.

While doing so, many of the irregularities and oddball statements of these languages were cleaned up, which helped it gain users (Bergin, 1996). The combination of features, input / output and solid mathematical features, made it a highly successful language. Pascal also improved the "pointer" data type, a very powerful feature of any language that implements it. It also added a CASE statement, that allowed instructions to branch like a tree in such a manner:

```
CASE expression OF
                    Possible – expression – value -1:
                        Statements to execute.....
                    Possible – expression – value – 2:
                        Statements to execute:
          END
```

According to Bergin (1996), Pascal helped the development of dynamic variables, which could be created while a program was being run, through the NEW and DISPOSE commands. However, Pascal did not implement dynamic arrays or groups or variables, which proved to be needed and led to its downfall. Wirth later created a successor to Pascal, Modula-2, but by the time it appeared, C was gaining popularity and users at a rapid pace.

C was developed in 1972 by Dennis Ritchie while working at Bells Lab in New Jersey. Verkeyn (2005) noted that C is a successor to B (which was developed by Ken Thompson at Bell Labs) and CPL was developed by the Universities of Cambridge and London; Martin Richards later developed BCPL (Basic CPL). Ferguson (2000) quickly added that the transition in wage from the first major languages to the major languages of today occured with the transition between Pascal and C. Its direct ancestor are B and BCPL, but its similarites to Pascal are quite obvious. All of the features of Pascal including the new ones such as the CASE statement are available in C. C uses pointers extensively and was built to be fast and powerful at the expense of being hard to read. But because it fixed most of the mistakes Pascal had, it won over former – Pascal Users quite rapidly.

Ritchie developed C for the new UNIX system being created at the same time. Because of this, C and UNIX go hand in hand. Unix gives C such advanced features as dynamic variables, multitasking, interrupt handling, forking and strong, low – level, input – output. Because of this, C is very commonly used to program operating systems such asUnix, Windows, the MacOS and Linux. Verkeyn (2005) submits that C was one of the most powerful languages of all times, used for almost all purposes (operating systems, wordprocessors, database, games, computer animations in movies etc) and that it was the first language designed by programmers which was very well structured, powerful, portable and flexible. However, it allows for very cryptic expression and dirty "pointer" hunting.

In the late 1970's and early 1980's a new programming method was being developed. It was known as Object Oriented Programming or OOP for short. Objects are pieces of data that can be packaged and manipulated by the programmer. Bjarne Stroustrop liked this method and developed extensions to C known as "C with classes". This set of extensions developed into the full – featured language C++ which was released in 1983 (Ferguson, 2000).

C++ was designed to organize the raw power of C using OOP, but maintain the speed of C and able to run on many different types of computers. C++ is most often used in simulations, such as games. C++ provides an elegant way to track and manipulate hundreds of instance of people in elevations, or armies filled with different types of soldiers. It is the language of choice in today's AP Computer Science courses (Ferguson, 2000).

In the early 1990's, interactive TV was the technology of the future. Sun Microsystems decided that interactive TV needed a special, portable (can run on many types of machines)language. This language eventually became Java. In 1994, the Java project team changed their focus to the week, which was becoming "the cool thing" after interactive TV failed. The next year, Netscape licensed Java for use in their internet browser, Navigator. At this point, Java became the language of the future and several companies' annonuced applications which would be written in Java, none of which came into use (Ferguson, 2000). Though Java has very lofty goals and is a textbook example of a good language, it may be the "language that wasn't". It has serious optimization problems meaning that programs written in it run very slowly. And Sun has hurt Java's acceptance by engaging in political battles over it with Microsoft. But Java may wind up as the instructional language of tomorrow as it is truly object – oriented and implements advanced techniques such as true portability of code and garbage collection (Ferguson, 2000).

Visual Basic often taught as a first programming language today as it is based on the BASIC language developed in 1964 by John Kemeny and Thomas Kurtz. BASIC is a very limited language and was designed for non – Computer Science people. Statements are chiefly run sequentially, but program control can change based on IF...THEN, and GOSUB statements which execute a certain block of code and then return to the original point in the program's flow (Ferguson, 2000)

Microsoft has extended BASIC in its Visual Basic (VB) product. The heart of VB is of the form or blank window on which a user drags and drop components such as menus, pictures and slider bars. These items according to Ferguson, (2000) are called "widgets". Widgets have properties (such as its colour) and events (such as clicks and double-clicks) and are central to building any user interface today in any language. VB is most often used today to create quick and simple interfaces to other Microsoft products such as EXCEL and ACCESS without needing a lot of code, though it is possible to create full applications with it.

Perl has often been described as the "duct tape of the internet", because it is most often used as the engine for a web interface or in scripts that modify configuration files. It has very strong text matching functions which make it ideal for these tasks. Perl was developed by Larry Wall in 1987 because the Unix Sed and awk tools (used for text manipulation) were no longer strong enough to support this needs. Depending on whom you ask, Perl stands for Practical Extraction and Reporting Language or Pathologically Electic Rubbish Lister (Ferguson, 2000).

Other programming languages include PL/I, Simula, Logo, Prolog, Smalltalk, Scheme, Ada, Objective – C, Eiffel, Object Pascal, Haskell and CLOS (Verkeyn, 2005).PL/I (Programming Language I) was developed around 1964. One of the goals of developing PL/I was to combine FORTRAN and COBOL (which turn out to be a monumental failure). It was better than COBOL but not as efficient as FORTRAN. The program was well structured for its time, but was overly complex for the moderate programmer (Verkeyn, 2005).

SIMULAtion was developed in 1967 by Ole – Johan and Kristen Nygaard. Simula was designed for simulation and modelling purposes, however, the language could be used as a general – purpose language. It was the first language with object oriented features but it was never really successful (Verkeyn, 2005). Logo was developed in 1968 by Seymour Papert at Massachusetts Institute of Technology (MIT). It was a versatile computer language that provides a friendly introduction to programming, a serious tool for advanced programmers and a medium for educational discovery. Verkeyn (2005) also noted another language; Prolog (that is PROgramming in LOGic) which was developed in 1972 by Alain Colmerauer (University of Aix - Marseille), Philipe Roussel (University of Edinburgh). It was a logic – language based on logical rewrite systems and used for automatic proof verification systems and other Artifical Intelligence Applications (Verkeyn, 2005). Smalltalk was introduced in 1972 at Xerox Palo Alto Research Center (PARC) by Alan Kay, Daniel Ingalls, Adele Goldberg, David Robson and others. It's an extreme Object Oriented language which is completely dynamic, interpreted (and thus rather slow) and allows for very fast programming (very well suited for prototyping). Guy Lewis Steels and Gerald Jay Sussman developed "Scheme" (another programming language) in

1975 which was a statistically and properly tail-recursive dialect of the LISP programming language designed to have an exceptionally clear and simple semantics and few different ways to form expressions (Verkeyn, 2005). Named after the first female programmer Lady Ada Lovelace, the programming language called “Ada” was based on the specifications of the USA department of Defence developed in 1983 by Jean Ichbiah, Bernd Krieg – Brueckner, Brian Wichmann and others. The goal was to develop a common, powerful language that could be used by all official (governmental) institutes of the USA. Ada was very well structured and powerful, with a lot of features that are missing from other languages (such as real – time, parallel and distributed systems). It is surprising that nowadays, Ada is still used in some governmental USA institutes (Verkeyn, 2005). “Object - C” was developed in 1986 by Brad Cox who wanted to add object oriented facilities to the popular C languages but based his features on the dynamic Smalltalk system. Objective – C could never compete with C++. “Eiffel” was conceived by Bertrand Meyer in 1986 but was never really successful (Verkeyn, 2005). Object Pascal was developed in 1986 by Apple Computer and Nicolaus Wirth. It adds object oriented features to Pascal. Infact the popular graphical developement environment Borland Delphi is based on this Language (Verkeyn, 2005). In 1987 “Haskell” (named after the mathematician Haskell Curry who developed the formal foundation of functional programming languages) was developed to create a standardized functional programming language with easy evaluation. In 2002, Haskell was the functional language on which most research was being performed (Verkeyn, 2005). Lastly, we shall consider CLOS in this section. CLOS (Common Lisp Object System)was developed by Daniel Bobrow, Sonya Keene, Linda De Michiel, Patrick Dussurd and others in 1988 which served as the standardised object oriented dialects of Common Lisp.

Programming languages have been under development for years and will remain so far many years to come. They got their start with a list of steps to wire a computer to perform a task. These steps eventually found their way into software and began to acquire newer and better features. The first major languages were characteristed by the simple fact that they were intended for one purpose and one purpose only, while the languages of today are differentiated by the way they are programmed in, as they can be used for almost any purpose; and perhaps the languages of tomorrow will be more natural with the invention of quantum and biological computers (Ferguson, 2000).

2.2.2 Historical Development of C++ Programming Language

The name C++ was coined by Rick Mascitti and it means C + 1 in the C-language. In addition, C++ was developed in 1985 by Bjarne Stroustrup at AT & T Bells Labs. An attempt to add object oriented facilities (from Simula) to the popular C language informed the development of C++ which was later standardised by ANSI/ISO, including a standard library (STL, Standard Template Library) (Verkeyn, 2005). C++ evolved from an earlier version called C with classes. The work and experience with C with classes from 1979 to 1983 determined the shape of C++.

The C++ programming language is basically an extension of the C programming language. The C programming language was developed from 1969 – 1973 at Bells Lab, at the same time the UNIX operating system was being developed there. C was a direct descendant of the language B which was developed by Ken Thompson as a systems programming language for the Fledgling UNIX operating system. B, in turn, descended from the language BCPL which was designed in the 1960s by Martin Richards while at MIT (Schildt, 1999; Berkakatin, 1995). In 1971 Dennis Ritchie at Bells Lab extended the B language (by adding types) into what he called NB; for "New B". Ritchie credits some of his changes to language constructs found in Algol68, although he states "although it (the type scheme) perhaps did not emerge in a form that Algol adherents would approve of " after restructuring the language and rewriting the compiler for B, Ritchie gave his new language a name "C" (Schildt, 1999; Berkakatin, 1995).

In 1983 with various versions of C floating around the computer world, ANSI established a committee that eventually published a standard for C in 1989. In 1983 Bjarne Stroustrup at Bell Labs created C++. C++ was designed for the UNIX system environment, it represents an enhancement of the C programming language and enables programmers to improve the quality of code produced, thus making reusable code easier to write (Schildt, 1999; Berkakatin, 1995).

Stroustrup (1995) outlines the history of the C++ programming language in his paper "A history of C++: 1979-1991". His approach of the historical developement of C++ emphasized on the ideas, constraints and people who shaped the language rather than the minutiae of language features. He traced the evolution of C++ from C with classes to the current ANSI and ISO standards work and the explosion of use, interest, commercial activity, compilers, tools, environment and libraries.

The prehistory of C++ (which refers to the couple of years before the idea of adding Simula – like features to C occured to him (Bjarne Stroustrup) is important because during this time the criteria and ideas that later shaped C++ emerged. He was working on his Ph.D thesis in the Computing Laboratory of Cambridge University in England. His aim was to study alternatives for the organisation of system software for a distributed system. The conceptual framework was provided by the capability – based Cambridge CAP computer and its experimental and continously evolving operating system. The details of this work and its outcome (Stroustrup, 1979) are of little relevance to C++. What is relevant, though, was the focus on composing software out of well – delimited modules and that the main experimental tool was a relatively large and detailed simulator he wrote for simulating software running on a distributed system. The initial version of this Simulator was written in Simula and ran on the Cambridge University computer center's IBM 360 / 165 main frames. The way Simula classes can act as co – routines made the inherent concurrency of his application easy to express. For example, an object of class computer could trivially be made to work in pseudo – parallel with other objects of class computer. Class hierarchies were used to express variants of application level concepts. For example, different types of computers could be expressed as classes derived from class computer and different types of inter – module communication mechanisms could be expressed as classes derived from class IPC. The use of class hierarchies was not heavy, though; the use of classes to express concurrency was much more important in the organisation of his simulator.

Stroustrup further noted that during writing and initail debugging, he acquired a great respect for the expressiveness of Simula's type system and the ability of its compiler's ability to catch type errors. The observation was that a type error almost invariably reflected either a silly programming error or a conceptual flaw in the design. The latter was by far the most significant and a help that he had not experienced in the use of more primitive "strong" type systems. In contrast, he found Pascal's type system worse than useless – a strait jacket that caused more problems than it solved by forcing him to warp his designs to suit an implementation – oriented artifact. The perceived contrast between the rigidity of Pascal and the flexibility of Simula was essential for the developement of C++. Simula class concept was seen as the key difference and ever since he submits that classes are the proper primary focus of program design. Link times for separately compiled classes were abysmal: it took longer to compile 1/30th of the program and link it to a precompiled version of the rest than it took to compile and link the program as a monolith. This, he believed to be a more a problem with the mainframe linker than with Simula, but it was still a burden. On top of that, the run – time performance was such that there was no hope of using the simulator to obtain real data. The poor run-time characteristics were a function of the language and its implementation rather than a function of the application. The overhead problems were fundamental to Simula and could not be remedied. The cost rose from several language features and their interactions: run – time type checking, guaranteed initialisation of varibles, concurrency support and garbage collection of both user – allocated objects and procedure activation records. For example, measurements showed that more than 80% of the time was spent in the garbage collector despite ever produced. Simula implementations are better these days (15 years later), but the order – of – ever magnitude improvement relative to systems programming languages still has not materialised.

To avoid terminating the project, he re-wrote the simulator in BCPL and ran it on the experimental CAP computer. The experience of coding and debugging the simualtor in BCPL was horrible. BCPL makes C looks like a very high level language and provides absolutely no type checking or run - time support. The resulting simulator did, however run suitably fast and gave a whole range of useful results that clarified many issues for me and provided the basis for several papers on operating system issues (Stroustrup, 1978; 1979b, 1981a). Stroustrup emphasized that his background in operating systems work and his interest in modularization and communication had permanent effects on C++.

In April, 1979, a work started in the Computing science Research Center of Bell Laboratories in Murray Hill, New Jersey. This work on what eventually became C++ started with an attempt to anlyse the UNIX Kernel to determine to what extent it could be distributed over a network of computers connected by a local area network. Two problems soon emerged: how to analyse the network traffic that would result from the kernel distribution and how to modularize the kernel. Both required a way to express the Modula structure of a complex system and the communication pattern of the modules. This was the kind of problem he had become determined never to attack again without proper tools. Consequently, he set about developing a proper tool.

In October, 1979 he had a pre-processor called Cpre, that added Simula – like classes to C running and in March 1980 this pre – processor had been refined to the point where it supported one real project and several experiments. His records showed that the pre-processor in use on 16 systems by then. The first key C++ library, the task system supporting a co-routine style of programming (Stroustrup, 1980b, 1987), was crucial to the usefulness of "C with classes" as the language accepted by the pre-processor was called in these projects.

During the April to October 1979 the transition from thinking about a "tool" to thinking about a "language" had occured, but C with classes was still thought of primarily as an extension to C for expressing modularity and concurrency. A crucial decision had been made, though. Even though support of concurrency and Simula – style simulations was a primary aim of C with classes, the language contained no primitives for expressing concurrency; rather a combination of inheritance (class hierarchies) and the ability to define class member functions with special meanings recognized by the pre – processor was used to write the library that stopped the desired styles of concurrency. There are many applications for which support for concurrency is essential but there is no one dominant model for concurrency support; thus when support is needed it should be provided through a library or a special purpose extension so that a particular form of concurrency support does not preclude other forms.

This, the language provided general mechanisms for organizing programs rather than support for specific application areas. This was what made C with classes and later C++ a general – purpose language rather than a C variant with extensions to support specialised applications. Later, the choice between providing support for specialised applications or general abstraction mechanisms has come up repeatedly. Each time the decision has been to improve the abstraction mechanisms.

An early description of C with classes was published as a Bell Labs technical report in April 1980 (Stroustrup, 1980a) and later in SIGPLAN notices. The SIGPLAN paper was in April 1982 followed by a more detailed Bell Labs technical report "Adding Classes to the C language: An Exercise in Language Evolution" (Stroutrup, 1982) that was later published in software: Practice and Experience.Stroustrup further stated that another major concern was to avoid restrictions on the domain where C with classes could be used. The idea (which was achieved) was that C with classes could be used for whatever C could be used for.

This implied that in addition to matching C in efficiency, C with Classes could not provide benefits at the expense of removing "dangerous or ugly" features of C. The alternative way of providing "safety" inserting run – time checks for all unsafe operations, was (and is) considered reasonable for debugging environments but the language could not guarantee such checks without leaving C with a large advantage in run – time and space efficiency. Consequently, such checks were not provided for C with classes, though C++ environments exist that provide such checks for debugging. In addition, users can and do insert run-time checks (assertions (Stroustrup, 1991) where needed and affordable.

C allows quite low – level operations such as bit manipulation and choosing between different sizes of integers. There are also facilities, such as explicit unchecked type conversions for deliberately breaking the type system. C with classes and later C++ follow this path by retaining the low-level and unsafe features of C. In contrast to C, C++ systematically eliminates the need to use such features except where they are essential and performs unsafe operations only at the explicit request of the programmer.

Stroustrup summarised the features provided in the initial 1980 implementation as classes, derived class, public/private access control, constructors and destructors, call and return functions, friend classes, type checking and conversion of function arguments. During 1981 three more features were added: inline functions, default arguments, overloading of the assignment operator.

Since a pre-processor was used for the implementation of C with classes, only new features, that is features not present in C, needed to be described and the full power of C was directly available to users. Both of these aspects were appreciated at the time. Having C as a subset dramatically reduced the support and documentation work needed. C with classes was still as a dialect of C. Furthermore, classes were referred to as "An Abstract Data Type Facility for the C language" (Stroustrup, 1980a) supports for object – oriented programming was not claimed until the provision of virtual functions in C++ in 1983 (Stroustrup, 1984a).

Clearly, the most important aspect of C with classes and later of C++ - was the class concept. Many aspects of the C with classes' class concept can be observed in Stroustrup (1980a).A class is a user defined data type. A class specifies the type of the class members that define the representation of a variable of the type (an object of the class), specifies the set of operations (function) that manipulate such objects, and specifies the access users have to these members.

2.2.3 Historical Development of Java Programming Language

According to the designing team of Aptech limited (2005), Java is a programming language popularly used to build programs that can work on the internet. Its primary features are that it is object – oriented and a cross platform language. By cross platform, they mean that the programs can run across several platfoms such as Microsoft Windows, Apple Macintosh, and Linux and so on. Java is not only used for stand alone applications and Net based programs but also to create consumer devices and accessories programs such as cellular phones, palm pilots and other gadgets.

In 1994, James Gosling (Sun Microsystems) introduced Java – a new programming language which before then meant 'Island' in Indonesia or a particular blend of hot drink (Verkeyn, 2005).Aptech limited (2005) notes that although Java's initial development began as early as 1991, it took sometime for the final working version to reach the market. The basic objective behind developing the language was to create software that could be embedded in consumer electronic devices. Efforts were taken to produce a portable, platform independent language and the result of this led to the birth of a new language. Java was initially called "oak" slowly but gradually it was found that the internet users had similar problems of portability and platform independence and were looking for software that could address these issues. The language was found to be small, secure and portable. Thus Java which was initially developed to cater for small – scale problems was found capable of addressing large-scale problems across the internet.

Carter(1997) reviewed the history of Java and traced it to 1991 when a group of Sun Microsystems engineers led by James Gosling decided to develop a language for consumer devices (Cable box, etc). They wanted the language to be small and use efficient code since these devices do not have powerful CPUs. They also wanted the language to be hardware independent since different manufacturers would use different CPUs. The project was code – named Green.

These conditions led them to decide to compile the code to an intermediate machine – like code for an imagimary CPU called a virtual machine (actually, there is a real CPU that implements this virtual CPU now). This intermediate code (called typecode) is completely hardware independent. Programs are run by an interpreter that converts the bytecode to the appropriate native machine code. Thus, once the interpreter has been ported to a computer, it can run any bytecoded program (Carter, 1997).

Sun uses UNIX for their computers so the developers based their new language on C++. They picked C++ and not C because they wanted the language to be object – oriented. The original name of the language was Oak but they soon discovered that there was already a programming language called oak, so they changed the name to Java (Carter, 1997).The Green project had a lot of trouble getting others interested in Java for smart devices. It was not until they decided to shift gears and market Java as a language for web applications that interest in Java took off. Many of the advantages that Java has for smart devices are even bigger advantages on the web (Carter, 1997).

According to “The history and Evolution of Java”printed by Mc – GrawHill; between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language. Bill Joy, Arther Van Hoff, Jonathan Payne, Frank Yellin and Tim Lindholm were key contributors to the maturing of the original prototype. The original impetus for Java was not the internet, instead, the primary motivation was the need for a platform- independent (that is, architecture - neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls. Many different types of CPUs are used as controllers. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target. Although it is possible to compile a C++ program for just about any type of CPU, to do so requires a full C++ compiler targeted for that CPU. The problem is that compilers are expensive and time – consuming to create. An easier and more cost – efficient – solution was needed. In an attempt to find such a solution, Gosling and others began work on a portable, platform – independent language that could be used to produce code that would run on a variety of CPUs under differing environments. This effort ultimately led to the creation of Java (http://books.mcgraw-hill.com) .

About the time that the details of Java were being worked out, a second and ultimately more important factor was emerging that would play a crucial role in the future of Java. This second force was of course, the WorldWide Web. Had the web not taken shape at about the same time that Java was being implemented, Java might have remained a useful but obscure language for programming consumer electronics (http://books.mcgraw-hill.com).

However, with the emergence of the World Wide Web, Java was propelled to the forefront of computer language design, because the web too, demanded portable programs (www.books.mcgraw-hill.com).

Most programmers learn early in their careers that portable programs are as elusive as they are desirable. While the quest for a way to create efficient, portable (platform - independent) programs is nearly as old as the discipline of programming itself, it had taken a back seat to other, more pressing problems.

By 1993, it became obvious to members of the Java design team that the problems of portability frequently encountered when creating code for embedded controllers are also found when attempting to create code for the internet. In fact, the same problem that Java was initially designed to solve on a small scale could also be applied to the internet on a large scale. This realization caused the focus of Java to switch from consumer electronics to internet programming. So, while the desire for architecture – neutral programming language provided the initial spark, the internet ultimately led to Java's Large – scale success (www.books.mcgrawhill.com) According to Carter (1997), there are currently two versions of Java, the original version of Java is 1.0. As at November 1997, most browsers only support this version. The newer version is 1.1 (in addition 1.2 is in beta). Only MS internet Explorer in 4.0 and Sun's Hot Java browsers currently support it. Carter (1994) notes that the biggest differences in the two versions are in the massive Java Class libraries. Unfortunately, Java 1.1 applets will not run on web browsers that do not support 1.1 (however it is still possible to create 1.0 applets with Java 1.1 development systems).

In summary, it is important to note that Java started out as a research project which began in 1991 as the Green project. The project was chartered to anticipate and plan for next wave of computing. The 'Green Team' determined consumer devices and computers would cover and also focused on TV set – top boxes and interactive TV industries. Research efforts on Java gave birth to a new language 'OAK' and were created by James Gosling (the father of Java). Java language was created with 5 main goals: it should be object – oriented, a single representation of a program could be executed on multiple operating systems, it should fully support network programming, it should execute code from remote sources securely, it should be easy to use. Oak was renamed in 1994 and Java was publicly released in May, 27, 1995. As a product, it was targeted at internet development and in general, it was marketed as the language to add dynamic features to the web, also known as Applets. Java had early support from companies like Netscape communications. (www.Develop/intelligence.com,2003-2007 Develop/intelligence LLC).

As mentioned earlier, Java derives much of its character from C and C++. Thus, it becomes interesting at this stage to consider the relationship that exists (if any between C++ and Java). The fact that Java derives much of its character from C and C++ is by intent. The Java designers knew that using the familiar syntax of C and echoing the object-oriented features of C++ would make their language appealing to the legions of experienced C/C++ programmers. In addition to the surface similarities, Java shares some of the other attributes that helped make C and C++ successful. (www.books.mcgraw-hill.com)

First, Java was designed, tested and refined by real, working programmers. It is a language grounded in the needs and experiences of the people who devised it. Thus, Java is a programmer's language. Second, Java is cohesive and logically consistent. Third, except for those constraints imposed by the internet environment, Java gives the programmer ful control. The quality of program outcome depends on the programmer. Thus, Java is not a language with training wheels but a language for professional programmers (www.books.mcgraw-hill.com..)

Although, Java and C++ are related, Java has significant practical and philosophical differences. While it is true that Java was influenced by C++, it is not an enhanced version of C++. For example, Java is neither upwardly nor downwardly compatible with C++ (www.books.mcgraw-hill.com..) Of course, the similarities with C++ are significant, a C++ programmer is usually comfortable with Java.

It is worthy of note that Java was not designed to replace C++. Java was designed to solve a certain set of problems. C++ was designed to solve different set of problems and both C++ and Java will coexist for many years to come.

As reviewed already, computer languages evolve for two reasons: to adapt to changes in the environment and to implement advances in the art of programming. The environmental change that prompted Java was the need for platform – independent programs destined for distribution on the internet. However, Java also embodies changes in the way that people approach the writing of programs (www.books.mcgraw-hill.com..)

Specifically, Java enhances and refines the object – oriented paradigm used by C++. Thus Java is not a language that exists in isolation. Rather, it is part of an ongoing process begun many years ago. This fact alone is enough to ensure Java a place in computer language history. Java is to internet programming what C was to systems programming: a revolutionary force that changed the world.

2.3 Self Efficacy and Achievement in Computer Programming

Jegede (2009a) conducted a study in the Engineering faculty of the Obafemi Awolowo University, Ile – Ife, Nigeria to verify whether some computing and programming related background variables will predict Java self efficacy. The results show that the analysis of variance of the multiple regressionsthat yielded an F – ratio of 19.821 which was significant. This implies that a combination of computing and programming background variables related to Java self efficacy of the engineering students. The multiple regression analysis on the relationship between the dependent variables (Java self efficacy) and the combination of the four independent variables shows that using the four independent variables to predict Java programming self – efficacy gives a coefficient of multiple regression of 0.545 and a multiple correlation square (R^2) of 0.297. These values are statistically significant at 0.05 level, which suggests that only 29.7 percent of the variance of Java self efficacy were explained by the combination of the four independent variables. In the same study, he made a further attempt to determine the relative power of each of the independent variables to predict Java self – efficacy of engineering students. The findings show that the number of programming courses taken and the average score in programming courses taken had t-values of 7.520 and 4.397 respectively. The values of Beta weights for the two variables are 0.469 and 0.272 respectively. These values are significant at 0.05 level of confidence which implies that the two variables contribute majorly to the prediction of Java self efficacy. From the values of Beta weights and t – ratios for each independent variable, it is clear that the number of programming courses offered had the highest impact in the prediction of Java programming self efficacy followed by the average score of the programming courses offered.

Ramalingan and Wiedenbeck (1998) present a 32 – item self efficacy scale for computer programming. This scale as well as the rest of the article is fequently referred to in recent studies concerning programming self efficacy (Askar and Daveport, 2009; Weindebeck, 2005; Weindeback, LabBelle and Kain, 2004).

In Ramalingam and Wiedenbeck (1998) study, the Self – Efficacy scale was handed out to 421 students in the beginning of a C++ programming course as a pre – test. The students were asked to rate their confidence in doing programming task uisng a Likert scale 1 (not all confident) to 7(absolutely confident). In the end of the course, the same scale was administered to the same student group as a post – test.

Ramaligan and Wiedenbeck (1998) assessed their scale and found it to be highly reliable with a score of 0.98. Looking at their results an increase in self – efficacy between the pre – test and post – test is found, especially among the students with initial low self efficacy. No substantial difference was found between males and females.

Since Ramalingan and Wiedenbeck (1998) studied C++ programming students, a closer look at the study made by Askar and Davenport (2009), including 326 Java programming computer engineering students, gives wider perspective on programming self efficacy.

Askar and Davenport (2009) developed an instrument assessing Java programming self – Efficacy from the self efficacy scale of Ramalingam and Wiedenbeck (1998). This test consisted of 32 stems and the reliability was even greater (0.99). The results show with a significant differencce between male and female respondents with males having higher self efficacy than their female counterparts. The overall self – efficacy score increased with the students'experience, frequency of computer usage as well as mother's and siblings'computer usage.

A close relationship could be established between the findings of Jegede (2009a) and Askar & Davenport (2009). Jegede (2009a) found that the number of programming courses already offered could predict Java programming self efficacy. The result obtained by Askar and Davenport (2005) appear to confirm the relevance of self – efficacy to the acquisition of Java programming skills.

It should be noted that self efficacy is specific to a certain activity. Therefore, a person may have high self – efficacy in one domain, such as gardening and low self – efficacy in another such as computer programming Ramalingan, LaBelle and Wiedenbeck, (2004). However, self effficacy can be investigated among related tasks, an example is trying to investigate the self efficacy across different programming languages.

Several research studies have been conducted on computer programming (Mc Namarah & Pyne, 2004; Bryne & Lyons, 2001; Begum, 2003; Fowler, Campbell, mcGill & Roy, 2002). Today, many industries are keen to accept as many graduates as the academic institutions can produce and there is an assumption that any bright student can be successful in computer programming. However, experience in the classroom would suggest that this is not true. Students who are proficient in many other subjects sometimes fail to achieve success in programming (Byrne & Lyons, 2001). This is because a number of factors affect computer programming performance. Taylor and Mounfield (1989) conducted a study on the predictors of computer programming performance among college students. They used gender, high school computer science and work to predict performance in computer science at the college. They found that prior exposure whether at the high school or college level is an important factor to students' success in computer programming.

2.4 Intrinsic and Extrinsic Factors

In a research conducted by Van der Westhuizen and Du Toit (1994) on the factors influencing job satisfaction among black female teachers in South Africa indicated that intrinsic factors played an important role in determining job satisfaction. In another study by Nhundu (1994), intrinsic and extrinsic factors played an important role as precursor to perceived job satisfaction among the population of teachers. Generally, studies grouping predictor variables into intrinsic and extrinsic factors seem to be very rare. In particular, computer programming related studies grouping predictor variables into intrinsic and extrinsic factors appear to be very very rare. This study has filled that gap. The variables grouped as intrinsic factors in this study are: gender, computer experience, computer ownership, mathematics background, Locus of control, background in C++, and numberof programming courses taken before entering Java programming class. Institutional type was taken as the extrinsic factors. These factors have each been linked with the dependent variables and reviewed in the sections that follow.

2.5 Computer Experiences, Self Efficacy and Achievement in Programming

Like computer Anxiety, there is little agreement in the literature on a precise definition of computer experience (Garland and Noyes, 2004). Some researchers define it by the number of years of computer use, while others state that it is the number of hours of usage per week.

Computer experience is the phrase used to refer to whatever exposure an individual has on computer whether in school, at home or anywhere. It is obvious that a student who already has prior knowledge of the computer will require less time to learn a programming language when compared to a computer novice who of course, will have to spend quality time learning how to effectively operate the computer effectively before mastering computer programming.

However this does not imply that students with a good computer experience will perform better or will have a better self efficacy in a programming course than those with little or no experience. This is an argument that has caught interest of many researchers over the years. It is interesting to note, however, that the findings on this issue has been consistent. The trend in self efficacy since 1987 has been fairly constant. Hill, Smith & Mann (1987) found a significant correlation between computer experience and self efficacy beliefs in computer courses (including programming) among a sample of 133 female undergraduates. They found that experience influenced bahavioural intentions to use computers indirectly through self efficacy. Thus, positive past experience with computers will increase self efficacy beliefs. Ertmer, Evenbench, Cennamo and Lehman (1994) found that although positive computer experience increased computer self efficacy, the actual amount of experience (i.e time on task) had no correlation with the self efficacy beliefs of undergraduate students. Houle (1996) found prior computer training to be significantly correlated with self efficacy in computer courses. More recent findings on computer experience include those of Hoskey and Maurimo (2010), Byrne & Lyons (2001), Hagan & Markhan (2000), Wilson & Shrock (2001), Askar and Davenport (2009), Doyle, Stamouli & Huggard (2005).

Doyle, Stamouli, and Huggard (2005) conducted a study on computer anxiety, self efficacy and computer experience among first year, second year, third year and fourth year students. They found that there was a correlation between self efficacy and computer experience, their study show the existence of a significant positive relationship which demonstrates that as the level of computer experience increases so does the level of self efficacy. Thus, students level of belief in their abilites increases as they progress with their course of study.

Research by Bandura (1986) showed that efficacy perceptions develop gradually with the attainment of skills and experience. Individuals form their self – efficacy beliefs by interpreting information primarily from their previous experience (Bandura, 1994b, 1995). In addition to mastery experience, self efficacy appraisals are partly influenced by vicarious experience of observing others perform similar tasks. Ramalingan, LaBelle and Wiedenbeck (2004) investigated the effects of self efficacy and mental models of programming. Their results showed that self – efficacy for programming was influenced by previous programming experience.

Askar and Davenport (2009) carried out a similar study among freshman engineering students enrolled on an introductory Java computer programming course at Bilkert University, Ankara, Turkey using an instrument assessing Java programming self efficacy developed from the computer programming self efficacy scale of Ramalingan& Wiendenbeck (1998). Their simple Regression analysis revealed that the number of years experience a student had with computers had a significant linear contribution to their self – efficacy scores. Their result also shows a tendency to gain self – efficacy in computer programming as the computer experience increases.

These results appear to confirm the relevance of computer experience to self efficacy in Java programming language skills and are in agreement with Bandura's theory.

Achievement in computer course (including computer programming courses) is another area that has attracted the interest of researchers. Of recent, researchers have been looking into academic achievement in computer programming (McNamarah & Pyne, 2004; Byrne & Lyons, 2001; Begum, 2003; Fowler, Campbell, McGill & Roy, 2002 and Erdogan, Aydin & Kabaca, 2007).Several factors have been shown to influence achievement in computer programming courses. Among factors such as gender (Cassidy and Eachus 2002; Jegede, 2007), mathematics or science background (Byrne & Lyons, 2001; Wilson & Shrock, 2001,Byrne & Lyons, 2001; Thomas, Woodbury & Jarman, 2002) and others, computer experience is most frequently mentioned (for example, Byrne & Lyons, 2001; Hagan & Markham, 2000; Wilson & Shrock, 2001). These studies provide converging evidence that computer experience has a positve effect on success in a programming course.

2.6 The Influence of Gender on Achievementand Self Efficacy in Computer Programming

Available literatures reveal that some variables have been identified as predictors of students' achievement in general compùting as well as in programming. Generally computers are identified with the areas of mathematics and science. It is commonplace that these areas are known for sex-related differences. Expectedly these sex-related differences had overtime been noticed in the computer technology disciplines.

Various studies had been reported on gender differences in general computing and various aspects of computing (programming inclusive).Nourbakhsh, Hammer, Crowley & Wilkinson (2004) carried out a study to investigate gender differences over a 7- week robotics course for high school students. Findings showed that girls were more likely to have struggled with programming than boys. Wilson (2002) conducted a study to determine factors that promote success in an introductory college computer science course and also to determine what, if any, differences due to gender exists. His model included twelve (12) possible predictors for success in a computer science course. The predictors were: mathematics background, previous programming experience, previous non programming experience, Attribution (Luck), Attribution

(Difficulty), Attribution (Effort), Attribution (Ability), Comfort level, Work preference, Self efficacy, Encouragement and gender. In any of the full – model significant variables identified in the study at the alpha level of 0.05, no signifcant differences between females and males were found. However, a significant difference between genders was found on playing games on the computer.

It then follows generally that in most of the studies the gender based differences are not significant. The differences noticed could be explained by reasons suggested by Authors like Bolan (2000). According to Bolan (2000), female disaffection from the disciplines traditionally identified as a precursor or gateway to a career in the field of IT (Mathematics and Sciences) begins as early as the 8th grade. Other factors that would also affect female participation and performance in computers were also identified as; (i) lack of female role models (Bolan, 2000); (ii) conceptualising their computer skills differently (Clegg & Trayhurn, 1999); and (iii) lack of a supporting academic computer science environment (Cohoon, 2001).

Owing to the reasons given above for the better performance of male students compared to their female counterparts, it has been hypothesised that the young age of participants and their limited cultural indoctrination regarding gender stereotypes would allow boys and girls to have equal success in programming related tests (Sullivan & Bers, 2012). Research findings has also suggested that children who are exposed to science, technology, engineering, mathematics (STEM) curriculum and programming at an early age demonstrated fewer gender-based stereotypes regarding STEM careers (Metz, 2007).

Sullivan & Bers (2012) carried out a study on a program named "The Tangible K Robotics Program". The study was meant to determine whether kindergarten boys and girls would be equally successful in a series of building and programming task. Kindergarten teachers were trained to implement the tangible K curriculum in their classrooms.

The training lasted for 3 hours. During the trainning, the teachers worked with a research assistant in order to learn how to use the programming language and robotics kits and completing each of the curriculum activities.

Also, while teaching, each of the teachers received technical and assessment support in their classrooms from research assistants. During each lesson in the curriculum, children were assessed by their classroom teachers or a research assistant. To ensure proper assessment, research assistants and teachers were both trained on administration of the assessment tools before the implementation of the study and no other adults were allowed to assess the children. Because of the uniqueness of the curriculum a new assement tool was created for the purpose of the study. Based on the data collected from assessments, classroom observations, interviews with students and teachers and analysis of students'work, the assessment tools developed for tangible K were refined several times to increase the face validity of the measurements based on teacher and researcher feedback.

Besides, outside consultants were also asked to evaluate and improve the measurements.
In addition to taking notes on children's key understandings and misconceptions, children were also, each assessed in small groups (approximately 4 children) on each child's achievement of the core goals of the activity. Children's learning achievement was determined based on conversation with the child during the activity, interview questions looking at what they built and looking at the programs they created. Children were assessed on the thoroughness of their understanding and application of core concepts and skills in each lesson using the Tangible K assessment form, on a 6 -Point likert scale, as follows:

5 – complete achievement of the goal, task or understanding;
4 – mostly complete achievement of the goal, task or unnderstanding;
3 – partially complete achievement of the gaol, task or understanding;
2 – very incomplete achievement of the goal, task or understanding;
1 – did not complete the goal, task or understanding;
0– did not attempt task.

Specifically, for programming, participants used the CHER P (Creative Hybrid Environment for Robotic Programming) program, the LEGO$^{(R)}$ brick from the LEGO$^{(R)}$ MINDSTROM^{TM} kit, and a variety of art materials to build and program their robots. CHER P, designed for the Tangible K Robotics Program, is a hybrid tangible / graphical computer language designed to provide young children with an engaging introduction to computer programming. The findings from the study showed that; although boys had a higher mean score than girls on more than half of the tasks, very few of these differences were statistically significant. Boys scored significantly higher than girls in two areas namely: properly attaching robotics materials and programming using IFS. Overall, both boys and girls were able to successfully complete the program.

Owing to the evidences from the literatures reviewed it is reasonable to agree with Sullivan and Bers (2012) that early introduction to programming and the proper orientation of the female students that programming is not solely a male domain, boys and girls would end up having equal success in programming related tests.Studies have indicated that many different factors can influence self efficacy in computer generally as well as in computer programming

Researches investigating the self efficacy beliefs in various tasks showed varied results with respect to gender. A study was conducted by Momanyi, Ogoma and Misigo (2010) on gender differences in self efficacy among science students in Lugari district in Kenya. One objective of the study was to investigate the influence of gender on self efficacy in science subjects among secondary school students. The participants responded to the items in a self efficacy questionnaire and the mean scores were computed. An independent sample t-test to compare the mean scores of male and female students showed that there was no significant difference between the self efficacy scores of boys and girls, ($t(228) = 0.36$, $p > 0.05$). It was therefore concluded that boys and girls in secondary schools do not differ in self efficacy in science subjects. The result of the study conducted by Momamyi,Ogoma and Misigo (2010) conducted among secondary school students agrees with that of Witt-Rose (2003) in a study conducted among college students enrolled in the course Anatomy and Physiology 1 (A & P) at Chipeira Valley Technical College (CVTC) in Eau Claire, Wisconsin in 2002. He found no significant relationship between gender and self efficacy.

This finding however contradicted that of previous reseachers. Past researches especially among secondary school students showed that girls had lower self efficacy than their female counterparts in science subjects. For instance De Backer & Nelson (2000) found that female students had lower self efficacy in mathematics and science compared to their male counterparts.Interestingly, Schunk & Lily (1984) found that gender differneces in mathematics self efficacy disappered when girls recieved clear performance feedback. Another study by Kenny – Benson, Pomerantz, Ryan & Patrick (2006) reported no gender differences in mathematics self efficacy.However, girls showed higher self efficacy in language arts than their male counterparts (Mecce, Ghenke & Burg, 2006).

Most of the studies on gender differences in computer self efficacy recoreded higher scores for male students than females. Cassidy and Eachus (2002) examined self efficacy beliefs in relation to computer use, experience and familiarity with a range of computer programs. The study was conducted among university students. The findings of that study was that men reported significantly higher computer self efficacy than women. The same study also reported that men were more experienced and familiar with more computer programs than the women. The study also showed that trainning received did not infleunece the trend in self efficacy scores reported as men still reported higher self efficacy than women even after the trainning they received.

Similarly, Durndell, Haag, Asemora & Laithwaite (2000) in a study found that male participants had higher computer self efficacy than females especially in advanced skills. Czaja, Charness, Fisk, Hertzog, Nair &Rogers (2006) also mentioned that women had lower computer self efficacy than men. Jegede (2007) in a study of factors affecting computer self efficacy among south – western Nigeria College of education lecturers however found that male lecturers (teacher educators) had a mean score of 85.51 while their female counterparts had 86.47. This showed a higher mean score in favour of the female lecturers. The study however showed that there was no significant difference between the computer self efficacy of male and female lecturers (teacher educators). Similarly, Busch (1995) studied gender differneces in self efficacy regarding complex tasks in word processing and spreadsheet software. The study found no gender differences in self efficacy regarding these computer tasks.

Computer self efficacy could be viewed from different perspectives. It is therefore a complicated concept with several determinants. Downey & McMurtrey (2007); Marakas, Johnson & Clay (2007) theorized that computer self efficacy is a multi-dimensional constructs that exists on several levels. In other words, computer self efficacy can exist at the levels of general computing and specific applications. Cassidy and Eachus (2002) noted that gender plays the most divisive role in people’s perceptions about complex technical tasks including programming skills.

According to Cassidy and Eachus (2002), the possible reason for this is that "the more complex the task is, the higher is the perceived masculinity factor, and hence men show higher self efficacy for such tasks".

Research reports on the relationship of gender and self efficacy in computer programming are scarce. The few available results however tend to show that men are more confident in computer programming than women. Wilson (2002) in a study looked at the relationship between 130 undergraduates' academic self efficacy and academic success. The primary focus of the study was to look at factors (self efficacy inclusive) associated with the academic success of female students compared to their male counterparts in a C++ programming course. The self efficacy factors were evaluated in terms of subject specific self efficacy. The computer programming self efficacy scale was administered to assess computer programming self effciacy of the participants in the study. Male students were reported to have higher programming scores when compared to their female counterparts. Also, in a study conducted by Vekiri & Chronaki (2008), the relationships between boys' and girls' computer experience, social support for using computers and motivational beliefs were examined. This was done to explore the possible gender differences in students' self efficacy and value beliefs. Results of the study showed that boys had more positive self efficacy and value beliefs about computers compared to girls and were more likely to engage in computer activities such as programming. Similarly, Busch (1995) found that males demonstrated higher perceptions in computer self efficacy and at the sametime had more experience with computer programming than females. Based on the strong relationship between experience with programming languages and computer self efficacy beliefs, gender differences in computer self efficacy may be attributed, in part to gender differences in experience with programming languages.

Askar and Davenport (2009) in a study conducted among 326 engineering students (200 first year students from the computer, electronic and industrial engineering departments and 20 first year science students, all of whom were enrolled in an introductory Java programming course, plus 106 second , third and forth year computer engineering students who volunteered to take part). The students were served a Java programming self efficacy scale along with a questionnaire consisting of demographic data including age, gender, department etc. The results of the study indicate that female students had significantly lower initial self efficacy beliefs compared with those of their male counterparts.

2.7 Computer Ownership, Self Efficacy and Achievement in Programming

Another factor found to have an impact on self efficacy and achievement in programming is computer ownership. Chilson, Carrey & Hemandez (2002) confirmed a positive effect of computer ownership on self efficacy. However Cassidy & Eachus (2002) in their study on the relationship between computer self efficacy, gender and experience with computers found that owning a computer was not a significant predictor of computer self efficacy.

Wilson (2000) studied factors that contribute to success in computer using computer ownership as one of the predictors. He found that computer ownership was the single most significant factor in course success for boys and girls at the 0.01 alpha levels. Only high school programming experience and owning a computer were significant in predicting success in computer science for males. A dramatic difference in the female success rate was shown for high school computer science courses where 30% more females who had taken the course succeeded compared to those who had not taken such a course.

Ogunkola (2008) conducted a similar research on computer attitude, ownership and use as predictors of computer literacy of science teachers in Nigeria. The subject for his study included one hundred and twenty science teachers drawn from the four political divisions of Ogun state in Nigeria. Two valid and reliable instruments named Computer Attitude, Ownership and Use Scale (CAOUS) and computer literacy self Assessment scale (CLASS) were used to collect the needed data, with a relaibility of 0.76 and 0.73 respectively through test – retest method of two weeks interval. Percentages, standard deviation and multiple regression statistics were employed in data analysis and findings reveal that a little above half of the science teachers had personal computers and not all the teachers used computer frequently. From his study, 14.3% of the variance in the teachers'computer literacy can be explained by the combined influence of the three variables.

The beta weights provide an indication of relative effects of each of the variables on the prediction of science teachers' computer literacy when other variables are controlled. The value t – ratio associated with the teachers'computer attitude and ownership are not significant at the 0.05 level but that of computer use is significant.

Although literatures are scare on the effects of computer ownership on computer programming self efficacy and achievement, yet this does not imply that computer ownership cannot be a predictor.On this ground computer ownership has been included as one of the variables in this study.

2.8 The Influence of Locus of Control on Self Efficacy And Achievement In Computer Programming.

In life, various things happen to individuals. Sometimes, they are pleasant and sometimes they are not. Individuals also see different people as responsible for whatever happens. Generally, according to Fakeye (2011) individuals have diverse beliefs about who controls his or her destiny. Some believe that ones destiny could be controlled by oneself, fate, God or powerful others.

Similarly students who pass or fail examinations tend to see either themselves or other people as responsible for their success or failure. The name given to this concept in literature is "Locus of control". Locus of control is an important aspect of psychology developed by Julian Rotter in 1966. Fakeye (2011) sees Locus of Control (LOC) as a generalised belief about the underlying causes of events of his or her life. Specifically, Araromi (2010) sees it as a sense of control. He therefore defines it as the extent to which an individual beliefs that he or she has control over an outcome. Rotter (1996) defines LOC as the degree to which a person belief that control of reinforcement is internal versus the degree to which it is external. Links have been found between Locus of control and behavior patterns in different areas. For instance, adults and children with an internal locus of control are inclined to take responsibility for their actions and are not easily influenced by the opinions of others. They also tend to do better at tasks when they can work at their own pace. By comparison, people with an external Locus of control (LOC) tend to blame outside circumstances for their mistakes and credit their success to luck rather than to their own efforts. They are readily influenced by the opinion of others and are more likely to pay attention to the status of the opinion holder, while people with an internal locus of control pay more attention to the content of the opinion regardless of who holds it. For instance; if a child with an internal Locus of control does badly in a test, she is likely to blame either her own lack of ability or preparation for the test. Whereas a child with an external Locus of control will tend to explain a low grade by saying that the test was too hard or that the teacher graded unfairly.

In the context of education therefore, locus of control refers to the type of attributions we make for our successes and/or failures in school tasks. Research has shown that having an internal Locus of control is related to higher academic achievement. The Locus of control construct is one of the most consistently researched variables in the social sciences (Lefcourt, 1992; Rotter, 1990). According to Rotter (1966) and Skinner (1996) the construct is based upon the principles from social learning theory and it captures people's general expectancies about the causes of rewards and punishments. It consists of two dimensions of causes: internal and external. Those with an internal Locus of control (LOC) generally expect that their actions will produce predictable outcomes. Those with an external Locus of control (LOC) generally expect that outcomes are due to external variables such as fate, luck or powerful others. Initially, the construct was conceptualized as uni-dimensional with internal and external Locus of control as opposite ends of a bipolar continuum. The "instrument of choice" for accessing adult's Locus of control across different situations has been Rotter's (1996). This was consistent with the conceptualization that internal and external Locus of control categories are mutually exclusive, this classic scale uses a forced- choice format.

However, the use of this scale in the 1960s and 1970s led to a series of inconsistent findings and consequently led to calls for revisions of this scale (Joe, 1971; Lefcourt, 1972). The result of these was the development of a multidimensional scale of control (Levenson, 1974). Levenson (1974) proposed that the inconsistencies in findings across research studies were not only due to the treatment of Locus of control as a unidimensional construct but also because there are actually two types of externals: (i) those who believe that the world is ordered and powerful others are in control; and (ii) those who believe that the world is unordered and events are due to non-human forces (such as chance, fate, luck etc.).

The Levenson (1974) scale which used the likert format (thus allowing the dimensions to be statistically used independently) the likert- format rather than a forced-choice format address the above concerns and thus became a common alternative to the standard Rotter scale. Furthermore, in the scale, the external dimension is categorized as either a belief that control is in the hands of human forces (i.e powerful others) or non-human forces (i.e chance). For those individuals who believe in powerful others, outcomes are unpredictable and control is not possible. Conclusively, according to Walkey (1979) and Skinner (1996), it is now generally accepted in the psychological literature that Locus of control is a multidimensional construct; and that Internal (I), powerful others (P) and choice (C) controls are theoretically independent constructs.

Amadi (2010) and Araromi (2010) in their studies posited that internal and external Locus of Control are important predictors for academic achievement. In paticular, Amadi (2010) is of the opinion that a more internal LOC is generally seen as desirable and relates to a higher academic achievement. According to Araromi (2010), those with a higher internal LOC belief that events results primarily from their own behaviour and actions; while those with a high/external LOC believe that powerful others, fate or chance primarily determine events. He also opined that those with a high internal LOC have better control of their behaviour, tend to exhibit more political behaviours and are more likely to attempt to influence other people than those with a high external LOC. They are also more likely to assume that their efforts will be successful and are also more active in seeking information and knowledge concerning their situation.

A number of previous studies have identified significant relationships between LOC and academic achievement. Stubbs (2001) in a study conducted concluded that internals tend to show superior achievement when compared to their external counterparts. Fakeye (2011) carried out a study among senior secondary school two (SSS2) students randomly selected from ten (10) secondary schools in Ibadan North Local Government. In that study, thirty (30) male and female students were randomly selected from each of the ten schools. In all three hundred (300) students participated in the study. He administered to the participants of the study, two (2) instruments namely; the Locus of Control scale and English Language achievement test. The Locus of Control scale was adapted from Araromi (2010) and was re – validated through a pilot survey. The reliability analysis using Cronbach alpha (r) yielded a co-efficient of 0.72. The English Language achievement test was adapted from West AfricanExamination Council (WAEC) paper two questions. It was revalidated using test – retest which gave the correlation coefficient value as 0.87. The data analysis used Pearson product moment correlation (PPMC) coefficient to determine the relationship between LOC and achievement in English Language. The T-test analysis was used to determine the significant difference in the English Language achievement of students with internal and external LOC. The result of the analysis revealed that there was a significant relationship between Achievement in English Language and LOC ($r = 0.670$, $p < 0.05$). The result of the analysis also showed that there was a difference which is not significant in the Achievement test in English Language of students with internal and External LOC ($t = 4.513$; $df = 298$, $P > 0.05$). The students with internal LOC performed better (mean = 41.4274) compared to their counterparts with external LOC (mean = 40.8295).

For sometime now, researchers have begun to examine the relationship between locus of control and information and communication technology in general and then specifically the relationship between Locus of control and Programming. More specifically, there has been a number of studies that examined relationship between Locus of control and programming skills. Bishop-Clarke (1995) identified personality traits such as Locus of control as factors that may help explain variability in Programming achievement.Jegede (2009a) opines that assessing the attribution classification of students and its relationship with programming skills might provide information that will facilitate better programming skills acquisition among students. Consequently Jegede (2009a) conducted a study using as participants 184 final year students in six departments of the Faculty of Technology of a University situated within the south western region of Nigeria. The departments included: Electrical and Electronic Engineering, Civil Engineering, Mechanical Engineering, Metallurgy and material Engineering, Chemical Engineering and Computer Engineering. In the study, it was found that programming achievement of engineering students has no significant relationship with the faith they have in their own life (internality; $r = 0.095$, $p \rangle 0.05$). Similarly, programming achievement of engineering students has no significant relationship with the belief in the irresistible power of others on their own lives (powerful others: $r = 0.068$, $p \rangle 0.05$). In addition the findings showed that the trust Engineering students place on chance in determining their own course of life is not a factor while considering programming achievement (Chance: $r = .017$, $p \rangle 0.05$).

In the same study the relationship between Locus of control and training initiatives in programming skills were studied. A significant relationship was found between internality and training initiatives in programming skills ($\chi^{2}_{1.184} = 0.043$, $p \angle 0.05$). The implication is that those with internal Locus of control do take the initiative of either registering for private training class in skills or undertake to learn those skills on their own as they perceive the needs. Also, significant relationship was not obtained between powerful others and training initiatives in programming skills among engineering students ($\chi^{2}_{1.184} = 0.069$, $p> 0.05$). Similarly, there was no significant relationship between those that believe in chance and their decision to embark on personal training on programming skills ($\chi^{2}_{1.184} = 0.733$, $p > 0.05$).

The studies on locus of control previously reviewed were mostly western based. Since, the society's belief also has cultural dimensions. There is the need to carry out the same study using the Nigerian participants. Jegede (2009b) that used the Nigerian respondents examined the relationship between the dynamics of locus of control on the programming skills of engineering students in a Nigerian University. Considering the challenges of epileptic power supply and lower accessibility to computers and software which may pose as challenges to continued personal development in programming skills, it becomes necessary to also look at the influence the locus of control of students have on their self efficacy in the relatively new object-oriented programming languages. Besides, since the participants in the previous studies were engineering students, it has become necessary to carry out an independent study using those who are supposed to be specialists in the field (the Computer Science students).

2.9 Mathematics Background, Self Efficacy and Achievement in Programming.

Mathematics background has, for a long time remained a predictor of programming ability (e.g Chung, 1988; Myloy & Burton, 1988; wilson, 2000).Wilson (2000) in a study of contributing factors to success in computer science found that mathematics background was the second (after comfort level) in importance in predicting success in the computer science class. The findings of the studies above confirm that mathematics is directly realted to programming performance ability. It has also been established in literature that high perceived self – efficacy could impact postively on performance and past performance as highlighted in Bandura's theory of self efficacy has the capability to increase perceived self efficacy in a particular task.

The link between mathematics and computer programming is widely accepted (Byrne & Lyons, 2001; Fowler et al, 2002). This may be becaue mathematics which has a fairly complex nature requires logical thinking and other features of computer porgramming. Thus the concepts which a student has to comprehend are similar to those in programming. Mathematics aptitude is thus often a pre – requisite for acceptance into computer science programs (Bryne & Lyons, 2001).

Hoskey and Maurino (2010) used mathematics and Logic background as one of the variables in their study on success factors for advanced programming. Previous programming experience and a mathematics background seem to be positvely related to success in introductory programming (Bryne & Lyons, 2001; Bennedson & Caserson, 2005; Wilson & Shrock, 2001; Rountreee, Rountree, Robins & Hannah, 2004). The subject used by Hoskey and Maurino (2010) have already completed two semester and had taken calculus and methods in operation research in mathematics (but this was not a pre – requisites for any of the programming classes, thus some students procrastrate and put it off).It was found that there was no significant difference between students who took calculus before the advanced Java course and students who took calculus after the Java course. Their result contradicts those obtained in the literatures already reviewed.

2.10 C++ Background, Self Efficacy and Achievement in Programming

Several studies have looked at the programming language used in the classroom. Of these, some analysed the programming languages for their teaching efficacy (Mannila, Peltomaki & Slakoski, 2006; Mannila & De Raadt, 2006) and others looked at the reasons colleges selected a particular programming language for an introductory programming course (Parker, Chao, Ottaway

&Chang, 2006). There was no conscensus on the best programming language to use. However, Mannila, Peltomaski and Salakoshi (2006) found that students did just as well learning a simple language and then moving on to a more complex one. They also found that the best languages to use in teaching programming were the languages designed with teaching in mind. They noted that a programming language is selected for many reasons beyond pedagogical benefit. In a study of employers and educators by Bhatnager (2009), the teaching of more than one language was recommended.

Hoskey and Maurino (2010) conducted a reasearch on success factors for advanced programming (Java programming language in particular) at Farmingdale State College where all students in the computer system Department in the school of Business were required to take two semester of programming at an introductory level. Students were offered a choice of C++ or Visual Basic and required to take an additional upper level programming course in Java. All students must achieve a “C” or better in both introductory programming classes to enter the advanced Java class.

Hoskey and Maurino (2010) had however observed that students entering the advanced class do not have the entry level programming skills needed to succeed in the upper level class. This may be due to the fact that students wait too long to take the advanced course and as a result, have forgotten what they learned in the introductory classes. Betterstill, students may not be able to cope with the change in programming language.

Hoskey and Maurino (2010) also noticed that although some students pass the introductory programming courses, yet they do not know how to program. Thus they were motivated to embark on a research investigating the predictors of advanced programming taking Java programming into consideration. They obtained records from the college to create a database containing information about all two hundred students who took Java porgramming language from 2005 through the fall of 2009 semester. It was found that students who took C++ for introductory programming classes were more successful than students who took Visual Basic for introductory programming classes using a one – tailed Mann – Whitney U test. Thus a student’s self efficacy and achievement in an advanced programming language (such as Java) may be predicted by the kind of introductory programming language he/she has been taught and mastered.

2.11 Number of Programming Courses Taken Before Java, Self Efficacy and Achievement in Programming.

In most academic institutions, Java programming language is not usually the first programming language. Jegede (2009c) in an attempt to examine the nature of programming preparatiom of trainee engineers conducted a study on computer programming curriculum of engineering undergraduates in a Nigerian University. His findings agree with the position of Schonberg and Davar (2008) that Java should not be introduced as an introductory programming course for the following reasons: Java hinders the understanding of code performance; the design methodologies of Java lead to a proliferation of objects, heavy use of dynamic storage and data structures that are pointer heavy and this considered wasteful; the model of concurrency in Java is low – level and error – prone and garbage – collected environment prevents its use in real – time application; the fundamental separation between specification and implementation is absent in Java. Wilson and Shrock (2001) investigated several factors (mathematics background, attribution for success / failure, domain specific work style preference, previous programming experience, previous non – programming experience amd gender) that can predict success in an introductory computer science course.

Their subjects included about 130 students who were enrolled in six sections of CS 202 Introduction to computer science at a Comprehensive Midwestern University during the spring of 2000. CS 202 was the first programming class required in the computer science major and uses C++ as the programming language. They used the computer programming selfefficacy Scale developed by Ramalingam & Wiedenbeck (1998) to collect data on domain – specific as it relates to tasks in C++ prorogramming language and reported an overall alpha reliability of 0.98 on the instrument. Programming experience which is a dichotomous variable determined by whether the subjects had engaged in any programming prior to the course was divided into the number of formal programming course taken and self – initiated programming experience. The result showed that the number of programming courses taken had a positive influence on midterm grade (programming achievement). Hoskey and Maurino (2010) found however, that there was no significant difference in the final Java grades for students who took more programming courses than required. Thus the number of programming courses already taken by some students did not contribute significantly to their success or achievement in Java programming course. This may be due to the factthat course taken (except Java) only serves to reinforce and reiterate materials already covered. Another explanation might be that students may have difficulty transfering the skills from one language to another (Hockey and Maurino, 2010). It is surmised that the logic course may help prepare the students for progamming, but not actually increase their programming ability (Hockey and Maurino, 2010).

However, Jegede (2009a) studied the predictors of Java programming self efficacy among engineering students in Obafemi Awolowo University, Ile-Ife, Nigeria by randomly selecting one hundred and ninety two final year engineering students randomly selected from six engineering departments of the University using programming Background Questionnaire and Java programming selfefficacy scale. The resulting data were analysed using Pearson Product Correlation and Multiple regression analysis. He found that Java programming self efficacy has no significant relationship with each of the computing and programming background factors. Jegede (2009) establishes that the number of programming courses offered by students and their achievement in the programming courses (based on scores) significantly predict their Java programming self efficacy. This appear consistent with the findings of Wiedenbeck (2005) who obtained that programming experience affected perceived self efficacy in programming and performance in programming courses.

In an earlier study, Ramalingan, La Belle and Wiedenbeck (2004) had come out with the results that self efficacy for programming was influenced by programming experience. Bandura (1986) also opined that self efficacy perceptions develop gradually with the attainment of skills and experience. The fact that self efficacy in programming domain becomes predictable by performance in programming course is logical. This is because learners with high self – efficacy are more likely to undertake challenging tasks and to expend considerably greater efforts to complete them in the face of unexpected difficulties, than those with lower self efficacy (Askar & Davenport, 2009).

However Jegede (2009a) argues that the number of years a student had been introduced to programming did not significantly predict Java self efficacy unlike the number of programming courses taken although both variables reflect programming experience. Jegede (2009a) explains that this may be understood in this way: experience in programming by year may not necessarily imply continuous active programming experience, for example, many of the engineering students in the various departments that he used for his study did offer for the first time programming courses in their second year. Apart from this, students might not just get involved in programming except in the semester during which programming as a course was compulsory, hence years of programming (which does not imply number of programming courses taken) did not predict Java self – efficacy.

2.12 Rationale for Multilevel Analysis

Investigating factors influencing students'self efficacyand achievement is complex. This is because some factors are peculiar to the students (Intrinsic factors) while others are outside students'control (extrinsic factors). Students (especially the subjects of this study) are trained by various institutions. Therefore, predicting variables of academic achievement and self efficacy at the universities can be viewed as a product of both the students' background variables (intrinsic factor) and their institutions (extrinsic factor). In this work, as in most educational researches, the population consisted of universities and undergraduates within the universities. In the sampling procedure, first we took a sample of institutions and then a sample of students in the computer departments of those schools. Therefore, it is clear that students are nested within the various institutions.

Specifically, we have student - level variables (gender, computer experience, locus of control, mathematical background, computer ownership, number of programming courses taken before entering the Java class and background in C++) describing individuals and the individuals are grouped into larger or higher order unit (institutions). This type of data nested within groups or collected at multiple levels simultaneously is known as multilevel data and the appropriate data analysis for them is known as the multi-level data analysis. In this instance, levelsrefers to how the data are organised and more important, statistically to whether observations are dependent or not(Nezlek, 2008). Normally, these levels are referred to by number (Level 1, level 2, level 3, etc) with larger numbers indicating levels that are higher in the hierarchy (Enders & Tofighi, 2007). For example, in this study, measures describing individuals would constitute the level 1 data, and measure describing the group (Institutions) would constitute the level 2 data. The number of levels in this work is limited to 2 because single classes have been chosen in each of the institutions. There are sets of data in educational settings that are collected in three or more levels. An example is a data of students nested within classes, classes nested within schools and schools nested within regions or countries.

In hierarchical data, individuals in the same group are also likely to be more similar than individuals in different groups. As a result, variations in outcome (e.g achievement in computer programming, self efficacy in Java programming) may be due to the difference between groups and to individual differences within the group. Therefore a model where disturbance may have both a group and an individual component can be of help in analysing data of this nature. This model known by various names in the literature (Multi-level models, hierarchical linear models, mixed–effects models, random effects models, random coefficient regression models, covariance component models) have led to new developments in the field of educational research.

According to Raudenbush and Bryk (2002), failure to consider the hierarchical structure of eductional data coud cause unreliable estimation of the effectiveness of school context or teacher quality on student learning outcomes and could misdirect educational policies and practices.Traditionally, the fixed parameter linear regression models are used for the analysis of such educational data which are hierarchical in nature and statistical inferences are always based on the assumptions of linearity, normality, homoscedasticity and independence. Because of the nested structure of the data in this work therefore, hierarchical linear modelling is important in analysing the data without bias (Akyuz, 2006). According to Raudenbush and Bryk (2002), not considering the multilevel structure of the data in the analysis, results in problems such as Aggregation bias, Misestimated standard errors and heterogeneity of regression.
Aggregation bias can occur when a variable takes on different meanings at different levels. For example, the institution of a student may have an infuence on a student's achievement above and beyond the effect of the students' related background factors. The multilevel analysis solves this problem by facilitating the decomposition of any observed relationship between variables into separate level – 1 (for students) and level – 2 (for institutions) component (Akyuz, 2006). Misestimated standard errors occur when the dependence among individual responses within the same organisation such as classroom or institution (as seen in this study) is not taken into account. The reason for the dependence may be shared experience of the individuals within the

classrooms and institutions or due to the sampling procedures. According to Akyuz (2006), hierarchical linear models solve this problem by incorporating into the statistical model a unique random effect for each institution. The variability in these random effects is taken into account in estimating standard errors. Heterogeneity of regression occurs when the relationships between individual characteristics and outcomes vary across organisation. This problem is solved by estimating a separate set of regression coefficients for each institution and then modelling variation among the institutions as multivatirate outcomes to be explained by institutional factors.

According to Wang and Bird, (2011), three major concerns should be taken into consideration when analysing multi – level data:(i) Unit of analysis (ii) Statistical procedures and (iii)Conceptual fallacy.

(i) **Unit of Analysis** – For most statistical analytic data procedures, the majorassumptions is independent observation (i.e students are independent from each other). But because students are somewhat alike within a particular classroom and somewhat different across classrooms, educational researchers often face the dilemma regarding what the unit of analysis should be. They are faced with the option of either analysing the data at the classroom / school level or the student level. When a teacher variable is involved, one way to examine the relationships between a student level variable and a classroom level variable is to aggregate student level variable with in each classroom (now using the classroom as the unit of analysis) and then correlate this aggregated means with teacher – level variables. According to Adams and Forsyth(2006) this method is limited because it reduces the statistical power significantly because, the sample size is now smaller. Besides, the procedure fails to consider within – classroom differences. Also the student – level variables aggregated to the classroom or school level are often highly correlated to each other and are likely to cause problems of multi-collinearity in regression analysis (Keeves & Sellin, 1990).

The second option opened to the researcher in examining these relationships is to conduct student – level analyses with all classroom level variables assigned to individual students (in this case using the student – level as the unit of analysis). It is obvious also that student within a particular calssroom is more like each other than they are like the students in any other classroom. Therefore the residuals involving student – level variables cannot be assumed to be independently and randomly distributed. Using the student- level as the unit of analysis therefore would be inappropriate for most commonly used inferential statistical tests. Additionally, by this method, the sample size for the classroom level variables is multiplied by the number of students in each calssroom and therefore the estimation of the coefficients between classroom level variables is likely to have a Type – 1 error (when an hypothesis is rejected when it should be accepted).

(ii) **Statistical Procedures**

According to Wang and Bird (2011), many statistical models have been developed for multi – level data analysis so that relationships between both student – level and classroom level variables can be analysed simultaneously and at individual and group – levels). Goddard, Tschannen – Moran and Hoy (2001) and Stapleton (2006) concluded that the use of a multi – level approach allows the researcher to examine relationships among variables within and between classrooms.

(iii) **Conceptual Falacy**

According to Hox (2002), a conceptual falacy occurs when researchers make interpretations of relationships at the individual level based upon aggregated data.Schwenkglenks (2007) gave the following as the potential evidence of ignoring hierarchical data in statistical analysis:

(i) A decrease in statistical efficiency and inflated standard errors may occur as no full use is made of the available information (Rice, 2001) Also, violation of the independence assumption can lead to false low standard error estimates (Hox, 2002). For both reasons, there is a risk of incorrect inferences regarding the existence of statistical associations and of incorrect decisions regarding the inclusion of exclusion of model parameters.

(ii) Effects on the response variables occuring at different hierarchical levels cannot be appropriately identified and explained (Hox, 2002). Related difficulties to interprete regression results may lead to an arbitrary choice of models and impact negatively as predictive ability (Woodhouse, Goldstein, 1989)

(iii) Variation of the response variable of interest occuring at the higher level (s) cannot be quantified at the population level, i.e any statement is at best possible for the higher level units directly observed (iv) Ultimately, wrong conclusions may occur (Aitkin, Anderson & Hinde, 1981).

In conclusion there are different techniques that reaserchers had used and are still using to address the problem of multi – level data structure in statistical analysis in research.
Schwenkglenks (2007) observed that most of the proposed techniques provide only partial solutions and suffer from sub–optimal use of the available information. For instance, some of them are suitable if the researchers'interest is restricted to either the lowest or the highest level in the hierarchy of observation and allow estimating unbiased standard errors. However according to Schwenkglenks (2007) a meaningful quantification of higher level variation in the total population of interest is only achieved by random effects analysis or multi – level modelling, and only the latter techniques is suitable to satisfactorily assess effects occuring at different levels or involving several levels (Austim, Goel & Walraven, Diez – Roux, 2002, Diez – Roux, 2000).

2.13 Appraisal of Literature Reviewed

Computer literacy has become a priority at all levels of our educational institutions today. In tertiary institutions for instance, a student before graduation must have offered and passed at least two computer courses. This is because, it is intended that all graduates are computer literate. Moreover, computer literacy is a requirement for employment in virtually all establishments nowadays. According to Kay (1989) and Van Dyke (1987), the knowledge and skills of a computer literate person may be divided into four basic categories; computer attitudes, computer application, computer systems and computer programming. Much has been studied about the first three categories. Programming which was not emphasized initially is now becoming more popular. This is because; people have now realized that programming is an important aspect of computing. Salomon and Perkins (1987) argue that programming provide an opportunity to develop rigorous thinking, learn the use of heuristics, nourish self consciousness about the process of problem solving, and in general achieve significant cognitive advances. Soloway (1993) further described programming as the new latin that enables learning in various other subject areas. Computers work with programs and so programmers are needed to develop more programs.

According to Mancy & Reid (2004), introductory programming courses are known for their notoriously poor pass rates. Such low pass rates indicate that students experience difficulties with programming. Many studies have been carried out to highlight the reasons for poor achievement in programming. According to Zimmerman and Schunk (2003)the predictive power of self efficacy beliefs on students' academic functioning has been extensively verified.

The predominant finding is that students' self efficacy beliefs are significantly and positively related to academic performance. Educational researchers recognize that, because skills and self efficacy beliefs are so intertwined, one way of improving students' performance is to improve student self efficacy (Wiedenbeck, Labelle & Kain, 2004). Several studies have found that self reported measures of computer experiences and software packages used are significant predictors of computer self efficacy (Hill, Smith & Mann, 1987); Torkazadeh and Koufterous, (1994) and Houle (1996) found prior computer training to be significantly correlated with computer self efficacy. In addition, Houle (1996) also found that high school courses with spreadsheet and databases and having worked at a job with computers contributed to higher computer self efficacy. According to Houle (1996), a prior high school programming class did not increase computer self efficacy.

Generally results of studies dealing with gender and self efficacy are inconsistent and the finding inconclusive. Boys are seen to report higher self efficacy than do girls in mathematics and science, whereas girls show higher self efficacy in language arts (Junge & Dretzke, 1995; Meece, Ghenke, & Burg, 2006; Siegle & Reis, 1998; Terwilliger & Titus, 1995). However, Schunk & Lily (1984) found that gender differences in mathematics self efficacy disappeared when girls received clear performance feedback. A more recent study by Kenny-Benson, Pomerantz, Ryan & Patrick (2006) reported no gender differences in mathematics self efficacy.

Research findings on the relationship of gender and self-efficacy in the computer programming are scarce. The few available literature, however, tend to show that men are more confident in computer programming than their female counterparts. Wilson (2002) in a study looks at relationship between 130 undergraduates' academic self-efficacy and academic success. The primary focus of the study was on the factors associated with the academic success of female students compared to male students enrolled in a C++ programming 1 course. The factors associated with academic success in the introductory programming course included previous computer experience, hostile environment and culture, attribution theory and self-efficacy.

The Locus of control construct is one of the most consistently researched variables in the social sciences (Lefcourt, 1992; Rotter, 1990). According to Rotter (1966) and Skinner (1996) the construct is based upon the principles from social learning theory and it captures people's general expectancies about the causes of rewards and punishments. It consists of two dimensions of causes: internal and external. Those with an internal Locus of control (LOC) generally expect that their actions will produce predictable outcomes. Those with an external Locus of control (LOC) generally expect that outcomes are due to external variables such as fate, luck or powerful others. Research has shown that having an internal Locus of control is related to higher academic achievement.

There have been a number of studies that examined relationship between Locus of control and programming skills. Bishop-Clarke (1995) identified personality traits such as Locus of control as factors that may help explain variability in Programming achievement. Similarly Chin and Zecker (1985) obtained that internals were more likely to succeed at program implementation than externals. Jegede (2009b) based on research findings opines that assessing the attribution classification of students and its relationship with Programming skills might provide information that will facilitate better programming skills acquisition among students.

Chung (1988) identified mathematics as a predictor of programming ability. Similarly Wilson (2002) in a study of contributing factors to success in computer science; mathematics background was found to be second (after comfort level) in importance in predicting success in the computer science class. The findings of the studies above confirm that mathematics is directly related to programming performance ability. It has also been established in literature that high perceived self-efficacy could impact positively on performance and past performance as highlighted in Bandura's theory of self efficacy has the capability to increase perceived self-efficacy in a particular task. Mc Coy & Burton (1988) study on the relationship of computer programming and mathematics in secondary school indicated a good mathematical ability as a success factor in beginners' programming.However literature on the influence of mathematical ability on perceived self-efficacy is scarce. This study will be investigating this influence.

Another factor that research has found to be impacting on computer self efficacy and achievement is computer ownership. Torkzadeh, & Koufterous (1994) and Houle (1996) in their studies found that owning a computer is found to be significantly correlated with computer self efficacy. Busch (1995) in a study carried out found that students who have access to their own computer cooperated more in front of the computer than any other group. Chilson, Carrey & Hemandez (2002) confirmed the positive effect of computer ownership on computer self efficacy. Cassidy & Eachus's (2002) study however showed that owning a computer was not a significant predictor of computer self efficacy.

2.14 Gaps of Literature Reviewed

This study was an attempt to bridge up deficit in research on computer programming achievement and self efficacy in Nigeria. Many researchers have studied on Achievement and self efficacy in areas different from computing and programming. Another major concern is that multilevel research on computer programming seems to be very rare generally all over the globe. In particular in Nigeria multilevel research reports appear to be very rare generally. Yet the Educational data we collect in most researchers are hierarchical in nature. Oftentimes, the fixed parameter multiple level regressions are used to analyse such data. In the process vital information are lost. This may lead to wrong conclusions. This study attempted to fill this gap.

CHAPTER THREE

METHODOLOGY

This chapter addresses the research design, population, sampling procedure and sample, instrumentation, procedure for data collection, method of data analysis as well as methodological challenges.

3.1 Research Design

The study was a correlational type of survey research. This is because it established relationships between predictor and criterion variables.

3.2 Variables In the Study

The variables involved in the study were as follows:

Predictor variables:

a. Computer Experiences
b. Gender
c. Mathematics Background
d. Computer Ownership
e. Locus of Control
f. Background in C++
g. Number of programming courses offered before entering Java class
h. Type of institution

Criterion variables:

The criterion variables are:

(i) Java Programming Self Efficacy
(ii) Java Programming Achievement.

3.3 Target Population

The target population for this study consisted of computer majors in Federal and state owned Universities in South–West, Nigeria. The choice of this group of students was based on the following assumptions:

1. Programming is an essential part of their training as computer majors.

2. Java programming is relatively new in Nigerian universities. Therefore the probability of getting respondents for the study among computer majors is higher than from Computer non – professionals.
3. Java programming is more dynamic and recent compared to some other programming languages offered by computer majors in Nigerian Universities.
4. The recommendations from the study will be most applicable to computer majors who by training have the option of becoming programmers rather than mere users.

3.4 Sampling Procedure and Sample

This study adopted purposive sampling for the selection of participants' universities and levels of study. The Universities of respondents were selected based on the following criteria:

(i) The university is owned by federal or state government

(ii) There is a computer science department where potential computer professionals are being trained

(iii) Java programming language is taught in the computer science department of the university.

In all, at the time of the study, five (5) Universities within the South – West, Nigeria satisfied the three criteria above. One of the five (Federal University of Technology, Akure, Ondo State, Nigeria) was used for the trial testing, validation and reliability of the instruments before the main study. The remaining four (4) Universities(University of Lagos, Obafemi Awolowo University, Ile-Ife, University of Ibadan and Adekunle Ajasin University, Akungba – Akoko, Ondo State, Nigeria) were used for the real study.

The levels of the participants for the study were selected based on the following criteria (i) the students at that level had been taught or are being taught Java in the previous or current semester, For those who are currently on it, they had covered enough ground to enable them answer the questions set. Different levels based on the programmes and peculiarities of each Universities were therefore used. At the university of Lagos, 200 level students who were almost through with the course participated in the study. The 200 level students were preferred because the present 300 level students were not taught Java when they were in 200 level. At the Obafemi Awolowo University, (O.A.U) the 400 level students were preferred and used for the study. At the O.A.U, the duration of computer science related courses is five years. Java programming language is taught at the 300 level. The year four students were taught in their second semester at the 300 level. At the time of data collection they just started the first semester in their 400 level. At the University of Ibadan, the course is taught during their second semester at the 200 level. The 300 level students who were in their first semseter were therefore preferred and used for the purpose of the study. The 400 level student in their first semester were preferred at the Adekunle Ajasin University Akungba Akoko (AAUA). This is because they were taught Java on their second semester year three.

Each participant used was selected based on the following criteria: (i.)he / she is a full time student in the Department ofComputer Science in any of the chosen Universities; (ii.) he / she had been taught Java programming Language; (iii.) he / she is available at the time of data collection and (iv.) he / she is patient and willing to participate in the study.All the students in the selected levels that were available weregiven the questionnaire. Some of the questionnaires were however not returned. Some that were returned were not properly filled. After scrutinisng the returned questionnaires only those that were properly filled were used for the study.

A total of 254 questionnaires and Java Programming Achievement Test question papers that were properly filled were used for the study.Table 3.1 shows the population of the levels sampled in each of the four universities and the number of completed questionnaire that were found useful

Table 3.1: The distribution of Respondents Across the Four Public Universities Used

S/N	Institution	Population	No of those given the Questionnaire	Returned questionnaire that were usable
1.	University of Lagos	85	82	68
2.	Obafemi Awolowo University, Ile Ife	129	111	76
3.	University of Ibadan	79	68	50
4.	Adekunle Ajasin University, Akungba-Akoko	120	60	60

3.5 Instrumentation

Instruments

Five instruments were used in the study. They were:

1. Computer Background Questionnaire (CBQ);
2. Computer Experience Scale
3. Java Programming Self efficacy scale (JPSES)
4. Java Programming Achievement Test (JPAT)
5. Levenson Locus of Control Scale (LLCS)

3.5.1. Computer Background Questionnaire (CBQ)

The Computer background questionnaire (CBQ) was used to obtain data on the biography of undergraduate computer students. Specifically, data on the following variables were collected with the use of the CBQ:

i. Computer Experience
ii. Gender
iii. Mathematics Background
iv. Computer Ownership
v. Background in C++
vi. Number of programming courses offered before entering Java class.

3.5.2. Computer Experience Scale

Computer experience is multidimensional. Different aspects of this variable is therefore highlighted in this work.Specifically the following 10 areas of computing were highlighted:Word Processing, Spread Sheet, Data Base, Presentation Software, Operating System Software, Computer Graphics, Computer Games, Internet, Statistical Package, Programming. The participants were given instruction to rate their experiences in the different areas of computing using a scale of 1 to 10. Maximum score obtainable is 100 while the minimum score is 10. The scale was also trial – tested on computer undergraduates at the Federal University of Technology, Akure, Nigeria (a group parallel to the subjects of the main study). The aim was to establish its reliability among computer undergraduates in South –West, Nigeria. The reliability coefficient using Cronbach Alpha was found to be 0.84. It was therefore found to be very reliable and fit for the study.

3.5.3 Java Programming Self Efficacy Scale (JPSES)

The C++ and Java programming languages have some similarities. Java programming language more or less grew out of the C++ language. Consequently the idea of C++programming self efficacy scale developed by Ramalingam and Windenbeck (1998) was used by Askar & Davenport (2009) to developed the Java Programming Self Efficacy Scale (JPSES). The JPSES was therefore the adapted version of the C++ programming Self Efficacy Scale of Ramalingam and Wiedenbeck (1998). The JPSES consisted of 32 items bothering on respondents' understanding of certain concepts in the language, possession and the ability to display certain skills as well as the will to persevere when writing the programs in Java. The participants were given instructions to rate their confidence in understanding and doing the Java programming related tasks using a scale of 1 (Not confident at all) to 7 (Absolutely confident). Maximum score obtainable is 224, while the minimum score is 32. Askar and Davenport (2009) administered it to Engineering undergraduates in Turkey who had been instructed in Java programming. They computed the JPSES scores and the reliability to be 0.99. The same instrument was trial tested on Computer undergraduates at the Federal University of Technology, Akure, Nigeria (a group parallel to the subjects of the main study) who had also been instructed on Java programming. The aim was to establish its reliability among Computer undergraduates in South-West, Nigeria. The reliabilty coefficient was found to be 0.96. It was found to be very reliable and fit to be used. The instrument as it is was therefore adopted for the study.

3.5.4 Java Programming Achievement Test (JPAT)

Achievement test in Java programming used for the study were of two types:

(a) Multiple choice questions in Java Programming (MCQJP) and

(b) Essay questions in Java programming (EQJP)

MCQJP was a 20 – item multiple choice objective questions selected and adapted by the researcher from a battery of questions available at http://india.com/java–programming and cs.dinke.edu/courses/fall 01/cps 001/quiz/online–questions. Questions were selected to cover three major topics in Java programming language that are being taught across the universities used for the study.They are: (1) Language fundamentals, (ii) Declaration and Access control and (ii) Arrays.

The researcher started with a pool of 94 questions. The draft copy of the pool of items was revised by two (2) computer science lecturers from two separate universities in the south –

western Nigeria each of which was given a copy of the draft question. Based on their reactions, sixty of the questions were selected for administration and item analysis. The sixty (60) items were administered to eighty (80) students in a university that was not part of the sample of the main study. Their responses to the items were scored;item analysis was carried out using discriminating power and difficulty index to collect the best 20 items that constitute the MCQJP (see Appendix V). The distribution of the 20 selected items was indicated in Table 3.2. The Reliability co efficient of 0.7was recorded for the MCQJP using KR – 20

A Table of Specification drawn for the final items selected to reflect the level of behavioural objectives; Knowledge, Comprehension and Application is shown in Table 3.2.

Table 3.2: Test blue print for the 20 multiple choice items on Java programming language

Content area	**Knowledge**	**Comprehension**	**Application**	**Total**
Language fundamental	1, 20, 5, 6	16, 17	12, 13, 14, 15, 18, 19	12
Declaration and access control	10	8, 9, 4, 7,		5
Arrays			3, 11, 2	3
Total	5	6	9	20

The Essay Question in Java Programming (EQJP) comprised of 2 programs originally written with some errors. The first one was written to compute the factorial of Numbers while the second one was written to compute the sum of two numbers (see Appendix). The respondents wereasked to identify the errors in the programs and then rewrite the codes for each of the lines with errors as indicated in the appendix. The programs were certified suitable by two lecturers in two of the Universities used for the study.

3.5.4. Levenson Locus of Control Scale

The Levenson Locus of Control scale was designed and validated by Levenson (1974). The Levenson scale conceptualized Locus of Control as multi-dimensional. The dimensions captured in the scale were: Internal (I), Powerful others (P) and Chance (C).Powerful others and chance were constructed as two components of externals. The scale has 24 items. Eight items each addressed each of the three dimensions. Each item is an attitude statement which represents a commonly held opinion to which the respondents indicated the extent to which they agreed or disagreed using the following response format:

Strongly agree: +3

Somewhat agree: +2

Slightly agree: +1

Slightly disagree: - 1

Somewhat disagree: -2

Strongly disagree: -3

The instrument was trial tested on computer undergraduates at the Federal University of Technology, Akure, Nigeria (a group parallel to the subjects of the main study). The aim was to establish its reliability among computer undergraduates in South – West, Nigeria. The reliability coefficient was found to be 0.88. it was therefore found to be very reliable and fit for the study. The instrument as it is was therefore adopted and used for the study.

3.6. Data Collection Procedure

The researcher engaged one research assistant who is very experienced in the administration of research instruments. The research assistant was trained on the administration of the instruments. He was also trained on how to handle the administration of the research instruments effectively. The research assistant handled the administration of the instrument in two universities (the university used for the pilot study and one of the four universities used for the main study). The researcher personally handled the administration of the research instruments in the remaining three (3) universities used for the main study. The data collection exercise lasted for four months. At the time of data collection, some of the universities were on vacation. The researcher had to wait for the schools to resume in order to get the computer undergraduates to respond to the instrument. This was responsible for the comparatively long period of data collection.

3.7 Scoring of the instruments

3.7.1 Computer Background Questionnaire

Gender was coded 1 and 2 for male and female respectively. **Ownership of computer** was coded 1 and 2 for yes and no respectively. For **Mathematics background;** the number of previous mathematics courses taken was coded using the number indicated by the respondents.**Background in C++** was simply coded 1 and 2 for Yes and No respectively.

3.7.2 Computer Experience Scale

Each of the ten (10) items under **Computer Experiences** was coded using the numbers from 1 to 10 (to indicate the level of experience) filled in the spaces provided in the front of the various computer experiences highlighted.

3.7.3 Java Programming Self Efficecy Scale (JPSES)

The coding pattern for all the 32 items that make up the JPSES

is as follows: Not at all confident = 1; mostly not confident = 2; slightly confident = 3; 50/50 = 4; fairly confident = 5 ; mostly confident = 6; absolutely confident = 7.

3.7.4 LevensonLocus of Control (MLOC).

The coding for the 24 items under MLOC scale is as follows:

Agree Strongly = +3

Agree Somewhat = +2

Agree Slightly = +1

Disagree Somewhat = -2

Disagree Strongly = -3

3.7.5 Multiple Choice Question on Java Programming (MCQJP)

Dichotomous scoring pattern of 1 and 0 for correct and incorrect responses was adopted.

3.7.6 Essay Question onJava Programming

Scores ranging from 0 to 5 were awarded for writing the correct codes. The maximum obtainable score is fifty four (54)

3.8 Data Analysis Procedure

The data collected was analysed using the Statistical Package for Social Sciences (SPSS) version 17.0 and Linear Structural Relations (LISREL) version 8.80 (Jöreskog & Sörbom, 2006). The following statistical procedures were used: Mean, Standard Deviation, Pearson Product Moment

Correlation (PPMC) Coefficient (Research Questions 1 and 2) and Multilevel Analysisusing LISREL 8.80 (Research Questions 3, 4, 5 and 6).

3.9 Methodological challenges

Below are some of the methodological challenges faced by the researcher in the course of this study.

3.9.1.1 The challenge of the Availability of literature

One of the challenges faced by the researcher during the preparation of the proposal is the dearth of empirical studies on the subject especially on studies carried out in Nigeria. After an extensive search, the researcher was able to discover only one Nigeria scholar who conducted a research that was not detailed on a small portion of the work. To solve this problem, the researcher was compelled to rely majorly on foreign authors for the review of literature.

3.9.1.2 Challenges from Sampling Techniques

Java programming language was adjudged the best for the study because of its advantages over other programming languages offered in the Nigerian Universities' computer curricula and its relationship with the internet. The researcher at the time of survey of the programming languages taught in the universities discovered that not all the universities have introduced Java programming language into their curricula. Five (5) (4 federal and 1 state) of the public Universities in South-West, Nigeria have Java in their curricula. Of the five (5), one of the federal universities was used for the pilot study while 3 federal and 1 state universities were used for the real study.

3.9.3 **Instrumentation challenges**

Generating multiple choice and essay questions in a programming language like Java that would be answered by students from different universities posed a challenge. To overcome this, the pool of questions adapted were taken to two of the universities (one state, one federal) situated in different states. There, the computer science lecturers validated the content. Based on their reactions to the pool of questions, sixty questions were generated and trial tested in one of the federal universities. From the item analysis carried out based on item difficulty and discrimination indices, the best 20 of the questions emerged for the real study.

CHAPTER FOUR

RESULTS AND DISCUSSION

This chapter presents the results and discussion of the findings. The presentation follows the order in which the research questions were presented in chapter one. The level of significance was set at p <0.05.

Research Question One: What type of relationship exists among intrinsic factors (gender, computer experience, computer ownership, Locus of control, background in C++, and numberof programming courses taken before entering Java programming class), extrinsic factor (type of institution)and computer undergraduates' Java programming self efficacy?

Table 4.1 presents the intercorrelation matrix of the correlation coefficients of the intrinsic factors (gender, computer experience, computer ownership, Locus of control, background in CPP, and numberof programming courses taken before entering Java programming class), extrinsic factor (type of institution)and Java programming self efficacy.

Table 4.1: Inter correlation Matrix of Intrinsic and Extrinsic factors and Self efficacy

Var	GD	MB	CE	CO	LOC	BCPP	NPCS	INST	JPSES
Gd	1.000								
MB	0.091	1.000							
CE	0.226*	0.232*	1.000						
CO	0.009	-0.225*	0.024	1.000					
LOC	-0.011	0.111	0.084	-0.131*	1.000				
BCPP	0.037	0.247	0.212*	-0.160*	-0.266*	1.000			
NPCS	0.030	0.245	0.154*	-0.181*	0.023	0.208*	1.000		
INST	0.010	.481*	.214*	0.260*	0.327*	0.628*	0.292*	1.000	
JPSES	0.104	0.270*	0.468*	-0.081	0.187*	0.350*	0.266*	0.431*	1.000
Mean	1.30	4.58	53.07	1.17	74.24	1.60	1.63	1.24	139.55
SD	0.46	2.40	14.98	0.38	20.97	0.49	1.05	0.26	44.15

*Note: Gd = Gender; MB = Mathematics Bachground; CE = Computer Experience; CO = Computer Ownership; LOC = Locus of Control; BCPP = Background in C++; NPCS = Number of programming course; JPSES = Java programming Self Efficacy Scores; INS Institution-level differences.. * p < .05*

From Table 4.1, it can be observed that in some, the relationship is positive and statistically significant at p < .05. For example, the relationship between Java programming self efficacy and computer experience is high and statistically significant (r = 0.468, p < 0.05). This relationship shows that as computer experience increases there is tendency for the undergraduate to have high computer programming self efficacy.The Table also shows that in some, the relationship is negative and also statistically significant. For example, the relationship betweennumber of programming courses and computer ownership is negative and statistically significant (r = - 0.181, p < 0.05).Computer ownership was coded 1, while non-ownership was coded 2. This means that there is a positive relationship which is very weak between ownership of computer and number of computer programming courses offered. Generally the highest relationship is between Java programmingself efficacy and computer experience (r = 0.468, p < 0.05), while the lowest is between locus of control andbackground in C++ (r = - 0.266, p < 0.05). The main criterion beingJava programming self efficacy, the Table shows that it has positive relationship with mathematics background (r = 2.270, p < 0.05), computer experience r = 0.468, p < 0.05, locus of control (r = 0.187, p < 0.05), background in C++ (r = 0.350, p < 0.05), and number of programming courses (r = 0.266, p < 0.05). However, Java programming self efficacy has the strongest relationship with computer experience.

Institution-level differences have significant relationships with mathematics background (r = 0.481, p < 0.05), computer ownership (r = 0.260, p < 0.05), background in C++ (r = 0.628, p < 0.05), Number of programming courses (r = 0.292, p < 0.05) computer experience (r = 0.214, p < 0.05), locus of control (r = 0.327, p < 0.05) and Java programming self-efficacy (r = 0.431, p < 0.05). Some of the relationships are weak, low and non-significant. For example, the relationship between mathematics background and gender (r = 0.091, p < 0.05) and that between Locus of control and computer experience (r = 0.084, p < 0.05) are both low and non-significant.

Discussion

Computer experience was found to have a positive and significant relationship with Java programming self efficacy (r = 0.468, p < 0.05). Thisfinding is in agreement with the findings of Koohang and Byrd (1987) as well as that of Taylor and Mounfield (1989). Jegede (2009a) however found out that number of years of programming experience and Java programming self efficacy were not significantly related.The inconsistency in the finding of this study and that of

Jegede (2009a) might have occured because of the differences in the measurement items and the respondents. Jegede (2009a) used number of years of programmingas the measure for computer experience; this study generated the measure of computer experience from the participants'self rating of their experience in ten areas of computing. Besides, Jegede's respondents were non computer majors (Engineering students) while this study used computer majors. For a non – computer major, longer years of programming experience may not be a good predictor of skill acquisition or self confidence in programming. Where there is no consistency in computer usage, years of computerusage might not be a good measure of computer experience. Also, consistency in utilisation of computer should be better among computer majors compared to their non computer major counterparts.

Another predictor that related positively and significantly with Java programming self efficacy is background in C++ ($r = 0.350$, $p < 0.05$). The implication is that those that had experience with CPP had higher self efficacy in their ability to write program using Java. This is expected as they are both object-oriented. The approach is quite similar. There is every likelihood that experience in C++ would boost their self efficacy in Java programming.

Mathematics background in this study was also found to have a relationship which is positive and significant with Java programming self efficacy. This finding could be because mathematics problem solving and programming require similar skills and ability to succeed. According to Harkins (2008), problem solving strategies employed in a traditional college mathematics course are essentially the same in a first course in computer programming.

Java programming self-efficacy was found to have very low and negative relationship with computer ownership. Ownership of computer was coded 1 while non-ownership was coded 2. The negative relationship therefore suggests that there is a positive relationship between ownership of computer and Java programming self-efficacy. The low relationship however show thatownership of computer does not necessarily affect Java programming self-efficacy of computer undergraduates. The implication is that those that own computer might have been occupied with carrying out activities other than Java programming on their computers.

The relationship between number of programming courses and computer ownership is negative and statistically significant (r = - 0.181, p < 0.05).Computer ownership was coded 1, while non-ownership was coded 2. This means that there is a positive relationship which is very weak between ownership of computer and number of computer programming courses offered. This then means that the relationship between computer computer ownership and offering of programming courses is very low.This might be explained by the poor achievement in computer programming which could in turn be a source of discouragement towards further participation in programming even when there is unrestricted accessibility to the computer.

The Institutional level differences was seen to relate positively and significantly with all the independent variables (except gender) and Java programming self-efficacy. The implication of this is that institution plays a very significant role in influencing undergraduates' intrinsic variables (such as mathematics background, computer experience, computer ownership, locus of control background in C++ and number of programming courses offered) and java programming self-efficacy. The institutional based factors should therefore be given attention when considering factors that could influence self-efficacy in Java programming.

Research Question Two: What type of relationship exist among intrinsic factors (gender, computer experience, computer ownership, Locus of control, background in C++, and numberof programming courses taken before entering Java programming class), extrinsic factor (type of institution)and Java programming achievement?

Table 4.2 presents the intercorrelation matrix of the correlation coefficients of the intrinsic factors (gender, computer experience, computer ownership, Locus of control, background in C++, and numberof programming courses taken before entering Java programming class), extrinsic factor (type of institution)and Java programming achievement.

Table 4.2: Inter correlation Matrix of Intrinsic and Extrinsic factors and achievement

Var	GD	MB	CE	CO	LOC	BCPP	NPCS	INS	JPAT
Gd	1.000								
MB	0.091	1.000							
CE	0.226*	0.232*	1.000						
CO	0.009	-0.225*	0.024	1.000					
LOC	-0.011	0.111	0.084	-0.131*	1.000				
BCPP	0.037	0.247	0.212*	-0.160*	-0.266*	1.000			
NPCS	0.030	0.245	0.154*	-0.181	0.023	0.208*	1.000		
INS	0.010	0.481*	0.214*	0.260*	0.327*	0.628*	0.292*	1.000	
JPAT	0.071	0.086	-0.023	-0.089	-0.046	-0.091	0.125*	0.058	1.000
Mean	1.30	4.58	53.07	1.17	74.24	1.60	1.63	1.24	21.10
SD	0.46	2.40	14.98	0.38	20.97	0.49	1.05	0.26	17.34

*Note: Gd = Gender; MB = Mathematics Bachground; CE = Computer Experience; CO = Computer Ownership; LOC = Locus of Control; BCPP = Background in C++; NPSS = Number of programming course; JPAT = Java programmingAchievement; INS Institution-level differences. * p < .05*

From table 4.2, it can be observed that it is only the relationship between Java programming achievement and number of programming courses that is positive and significant at p < 0.05; others are not statistically significant. Institution-level differences are significant with mathematics background, computer owneship, background in C++, Number of programming courses before entering Java programming class, computer experience and locus of control. It is not significant with gender and Java Programming achievement.

Discussion

The number of programming courses taken before entering the Java programming class has a weak, positive relationship which is statistically significant with Java programming achievement. This suggests that the computer undergraduates that took more courses in computer programming tend to perform better in Java programming achievement testswhen compared to their counterparts that took fewer courses in computer programming. This finding supports the findings of Taylor and Mounfield (1989). According to them, prior experience in programming provides a significant predictor of students' performance in programming courses. They submitted that students'success in computer programming is influenced significantly by their exposure either at the high school or college level.There is therefore the probability that computer undergraduates' exposure to variety of programming courses would enhance performance in Java programming. This is expected as there are some similar skills and procedures in the different programming languages. It is therefore reasonable to believe that repeated exposures to different programming languages would enhance both the understanding and performance in new programming languages to be learnt.

Research Question Three: How much of the total variance in Java programming self efficacy of computer undergraduates is accounted for by institution-level and student-level differences?

To answer this research question, a multilevel analysis was conducted with ordinary least square option of LISREL. The model used is known as null model in that only the intercept of institution-level (macro level) was entered. That is the focus was on variance decomposition of programming self efficacy on the basis of student-level differences (level 1) and institution-level differences (level 2).

Statistically, the null modelling for variance decomposition of self efficacy is given by

Self efficacy$_{ij}$ = $\beta_0 + u_{0i} + e_{ij}$

Where self efficacy$_{ij}$ represents score "j" for student "i"; β_0 represents the intercept of the fixed part of the model andu$_{oi}$ represents the random variation in intercepts at level – 2 of the model and eij denotes the random variation at level – 1 of the model.Table 4.3and 4.4 present the fixed part of the model and random part of the model.

Table 4.3: Fixed Part of the Null Model for computer programming self efficacy

Co-efficients	**BETA-HAT**	**STD.ERR**	**Z – VALUE**	**PROB**	**MODEL-FIT**
Intercept	13.21	2.96	4.67	.001	2107.10

The results in Table 4.3 indicates that the intercept of the fixed part ofthe model is 13.21 and it is statistically significant at p< .05. The model fit statistics given by -2 log – likelihood = 2107.10, p <.05 indicates that the null model adequately fit the data.

Table 4.4: Random Part of the Null Model for computer programming self efficacy

LEVEL	THAU – HAT	STD.ERROR	Z – VALUE	PROB
LEVEL2 Intercept / intercept	22.39	21.36	1.04	0.294
LEVEL 1 Intercept / intercept	228.94	20.44	11.20	0.001

We estimate the total variance in Java programming Self efficacy byusing equation 4.1 below:

$\frac{\Phi_1}{\Phi_2+\Phi_1}$= Total Variance ....(4.1)

Where $\Phi_2$represent between group variability (Variance of level-2) and $\Phi_1$represent within group variability (variance of level-1).

From table 4.4, Φ_2 = 22.39 and Φ_1 = 228.94

:. Total variance in Programming self efficacy = $\frac{\Phi_1}{\Phi_2+\Phi_1}$......................................(4.2)

This is given as $\frac{228.94}{22.39+228.94} = \frac{228.94}{251.33}$ = 0.91

Therefore the total variance in Java programming self efficacy accounted for by institution-leveldifferences is 0.91. This result indicates that about 91.0% of the total variation in programming self efficacy is explained by the differences ininstitution.

To obtain total variance accounted for by student-level differences (level 1) we use the formula.

Micro level differences for null model = 1 – Total variance at level 2

This gives

1 – 0.91 = 0.09

Therefore total variance accounted for by student-level differences is about 9.0%.

Discussion

The results of this study indicated that 91.0% of the total variation in programming self efficacy was accounted for by institution–level differences. The remaining 9.0% was accounted for by the student-level differences.Institutional – level differences contributed more to variation in programming self efficacy in this study compared to student level contributions. It then follows that institution – level differences play a significant role in determining the confidence computer undergraduates have in their ability to program (otherwise called self efficacy in computer programming).This claim could be supported by the finding of Gisemba (2011) in a study carried out to determine the extent to which teacher self efficacy and students mathematics self efficacy could enhance secondary school students' achievement.In that study, it was found that teachers frequent use of mathematics homework and level of interest and enjoyment of mathematics as well as their ability and competence in teaching mathematics played a key role in promoting students mathematics self efficacy.

To boost students' self efficacy in computer programming, students should not only be the focus, the institutions must also create an enabling environment that encourages and simplifies the learning of Java programming.It is reasonable to think that teachers that are part of the system, by using home work often, carrying out activities that would raise their level of interest and making Java programming very interesting are likely to increase their self-efficacy level in Java programming as revealed from the findings of Gisemba (2011).The learning environment for Java programming also includes the laboratory. A laboratory equipped with computers that contains the compilers of Java and that of other programming languages affords students access to practice writing programs as well as opportunities to see others write programs. Seeing others write correct programs (vicarious experiences) is a source of self efficacy. A good laboratory equipped with compilers of programming languages including that of java where students have access and also see others write programs becomes an enabling environment for the learning of Java.

Research Question Four: How much of the total variance in Java programming achievement of computer undergraduates is accounted for by institution-level and student-level differences?

To answer this research question, a multilevel analysis was conducted with ordinary least square option of LISREL. The model used is known as null model in that only the intercept of institution-level (macro level) was entered. That is the focus was on variance decomposition of programming achievement on the basis of student-level differences (level 1) and institution-level differences (level 2).

Statistically, the null modelling for variance decomposition of achievement is given by

$\text{achievement}_{ij} = \beta_0 + u_{0i} + e_{ij}$

Where achievement_{ij} represents score "j" for student "i"; β_0 represents the intercept of the fixed part of the model andu_{oi} represents the random variation in intercepts at level – 2 of the model and eij denotes the random variation at level – 1 of the model.Table 4.5and 4.6 present the fixed part of the model and random part of the model.

Table 4.5: Fixed Part of theNull Model for computer programming achievement

Co-efficients	BETA-HAT	STD.ERR	Z – VALUE	PROB	MODEL-FIT
Intercept	21.10	1.09	19.39	.001	69366.65

The results in Table 4.5 indicate that the intercept $\beta_o$of fixed part of the null model is 21.10 and it is statistically significant at p< .05. The model fit statistics given by -2 log – likelihood = 69366.65, p <.05 indicates that the null model adequately fit the data.

Table 4.6: Random Part of the Null Model for computer programming achievement

LEVEL	THAU – HAT	STD.ERROR	Z – VALUE	PROB
LEVEL2 Intercept / intercept	61.49	0.83	74.01	0.001
LEVEL 1 Intercept / intercept	273.67	0.09	3065.86	0.001

We estimate the total variance in Java programming achievement byusing equation 4.1 below:

$\frac{\Phi_1}{\Phi_2+\Phi_1}$ = Total Variance ....(4.1)

Where $\Phi_2$represent between group variability (Variance of level-2) and $\Phi_1$represent within group variability (variance of level-1).

From table 4.6, Φ_2 = 61.49 and Φ_1 = 273.67

:. Total variance in Programming achievement = $\frac{\Phi_1}{\Phi_2+\Phi_1}$.......................................(4.2)

This is given as $\frac{273.67}{61.49+273.67} = \frac{273.67}{335.16} = 0.82$

Therefore the total variance in Java programming achievement accounted for by institution-leveldifferences is 0.82. This result indicates that about 82.0% of the total variation in programming achievement is explained by the differences ininstitution.

To obtain total variance accounted for by student-level differences (level 1) we use the formula:

Micro level differences for null model = 1 – Total variance at level 2

This gives

1 – 0.82 = 0.18

Therefore total variance accounted for by student-level differences is about 18.0%.

Discussion

The result indicated that 82.0% of the total variance in programming achievement is explained by the differences in students'institutions. The remaining 18.0% is accounted for by the student level differences. Institutional – level differences contributed more to variation in programming achievement compared to the contributions of student-level differences.Studies that employed multilevel analysis of factors influencing students'achievement in programming are very rare. Specifically none was available to the researcher. However, there are several studies that were conducted to answer the level of contribution of institutiional – differences to achievement in mathematics. Research findings had consistently shown that mathematics ability is a good predictor of performance in Computer programming (Byrne & Lyons, 2001; Wilson & Shrock, 2001). Besides Erdogan, Aydin, and Kabaca (2007) submitted that computing as a subject requires a structure and appproach with which students have same experience and similar cognitive skills used in the study of mathematics. Since there seems to be a dearth of literature from studies on Computer programming, an attempt was therefore made to compare the results of this study with the similar ones done with mathematics.

Proportion of variance in achievement due to institutional – differences varies from one place to another. Park and Park (2006) found that in SouthKorea, about 4% of the total variance of mathematics achievement was due to institutional – level factors. For south african students it was 55% (Howie, 2006). Similarly for Australian students 27% and 47% formed the percentage contributions of institutional – level differences in the Trends in mathematics and science study (TIMSS) conducted in 1995 and 1999 respectively (Fullarton, 2004). In Singapore 45%, Botswana 27%, Chiki 30% and Flenders 14% were the percentage contributions found (Chepete, 2008, Mohammadpour, Maradi & Najid, 2009; Ramirez, 2006; VandenBrock, VanDamme, Opdenkker, 2006). Recently, a study carried out by Ghagar, Othman, MohammadPour (2011) among eight graders in Malaysia indicated that 57.28% of the total variance in mathematics achievement of eight graders in Malaysia was accounted for by institutional level differences. Also Stemler (2001) studied school effectiveness in mathematics and science at the 4th grade, using data from International Educational Assessment (IEA's) TIMSS study. 14 countries were included in the study. In general, about one – quarter (25%) of the variability in mathematics and science achievement was found to lie between schools.

A key result from the findings of this study and the previous studies reviewed is that institutions make a difference in students' achievements whether in mathematics or computer programming. The level of influence however differs. Comparatively, the percentage influence of institution – level differences in this study appears to be the highest when compared with the other studies in mathematics. In some of the previous studies like the ones conducted in South-Africa (Howie, 2006), Australia (Fullarton, 2004), Singapore (Chepete, 2008) and Ghagar et al (2011); the percentage influence were lower. The percentage influences of institution-level differences in those studies ranged between 27% and 57.2% while the one from this study was 82.0%.

The possible explanation for this could be the style of curriculum implementation in the Nigerian tertiary institutions including the institutions studied. In the Universities studied, lecturers only have access to the synopsis of the courses. The detailed specifications of the topics and the extent of coverage are not specified. The lecturer in charge is therefore left to use his discretion to determine the deatils of the content to be covered as well as the style of instruction. Since individuals are always unique in the way they do things, it is reasonable to think that the opportunities available to the different groups of students in the various schools studied and their experiences in the Java programming class differed. Differencesin the experiences could be the cause of the institutional level differences discovered in the present study. Unlike in the present study, the participants of the previous studies were Basic School students. At the Basic level of education, curriculum given to the teachers are more detailed. The content of what to be taught are well spelt out. Uniformity of content and general implementation of the curriculum is relatively uniform at this level unlike at the tertiary level.

Research Question Five: How much of the student-level variance in Java programming self efficacy of computer undergraduates is associated with gender, ownership of computer, mathematics background, background in C++, computer experience, LOC and number of programming courses taken before entering Java programming class.

To answer this research question, a multilevel analysis was conducted with ordinary least square option of LISREL. The model used is known as model 1in that explanatory variables of student-level differences plus the intercept of institution-level (macro level) were entered. That is, the focus was on linear growth model with random intercepts and their slopes

Statistically, the null modelling for variance decomposition of self-efficacyis given by

Self efficacy$_{ij}$ = β_0 +β_1 X Gd$_{ij}$ +u$_{0i}$ +u$_{ij}$ XGd$_{ij}$+ .... +β_nX NPCS$_{ij}$ +u$_{0i}$ +u$_{ij}$ XNPCS$_{ij}$e$_{ij}$

Where self efficacy$_{ij}$ represents score “j” for student “i”; β_0 represents the intercept of the fixed part of the model andu$_{oi}$ represents the random variation in intercepts at level – 2 of the model and eij denotes the random variation at level – 1 of the model,u$_{ij}$ represents the random variation in slopes for each of the intrinsic factors.Table 4.7and 4.8 present the fixed part of the model and random part of the model.

Table 4.7: Fixed Part of the Linear Growth Model 1 of Intrinsic factors

COEFFICIENTS	BETA-HAT	STD.ERR	Z-VALUE	PR > |Z|
Intcept	4.08	8.68	0.47	0.641
Gender	2.87	2.18	1.32	0.191
Owncomp	2.39	2.71	0.88	0.377
MBCGUNI	0.43	0.44	0.98	0.328
CPPBGD	3.31	2.19	1.52	0.130
NPGRMCS	1.15	0.59	1.96	0.050
COMPEXP	-0.03	0.07```````	-0.49	0.627
LOC	-0.04	0.05	-0.75	0.452

DEVIANCE= -2*LOG (LIKELIHOOD) = 53228.78

NUMBER OF FREE PARAMETERS = 45

Table 4.7 shows that out of the seven intrinsic factors considered, only number of programming courses taken by the computer undergraduates significantly contributed to the prediction model that is undergraduates self efficacy.Other intrinsic factors do not contribute significantly to the prediction model 1

Table 4.8: Random Part of the Linear Growth Model 1 of Intrinsic Factor

LEVEL 2	TAU-HAT	STD-ERR	Z-VALUE	PR >|Z|
Intercept/Intercept	-111.46	1.83	-60.91	0.001
Gender/Intercept	24.19	0.94	25.65	0.001
Gender/Gender	-15.75	0.90	-17.55	0.001
Npgrmcs/Intercept	10.77	0.91	11.79	0.001
Npgrmcs/Gender	0.86	0.61	1.41	0.160
Npgrmcs/ Npgrmcs	-0.60	0.82	-0.73	0.465
Owncomputer/Intercept	-14.35	0.97	-14.83	0.000
Owncomputer/Gender	11.88	0.67	17.61	0.000
Owncomputer/Npgrmcs	-2.68	0.65	-4.13	0.001
Owncomputer/Owncomputer	-6.64	1.00	-6.63	0.000
Mbcguni/Intercept	6.44	0.91	7.07	0.000
Mbcguni/Gender	-0.64	0.61	-1.05	0.293
Mbcguni/Npgrmcs	0.75	0.58	1.29	0.198
Mbcguni/owncomputer	-3.48	0.65	-5.36	0.000
Mbcguni/ Mbcguni	-0.98	0.82	-1.19	0.233
Cppbgd/Intercept	62.88	0.99	63.35	0.000
Cppbgd/Gender	-6.16	0.66	-9.33	0.000
Cppbgd/Npgrmcs	4.62	0.63	7.30	0.000
Cppbgd/owncomputer	-20.88	0.71	-29.51	0.000
Cppbgd/Mbcguni	4.04	0.63	6.38	0.000
Cppbgd/Cppbgd	-14.04	0.96	-14.56	0.000
Compexp/Intercept	0.34	0.91	0.37	0.710
Compexp/Gender	-0.33	0.61	-0.54	0.590
Compexp/Npgrmcs	-0.06	0.58	-0.11	0.912
Compexp/owncomputer	0.14	0.65	0.22	0.827
Compexp/Mbcguni	-0.00	0.58	-0.00	0.998
Compexp/Cppbgd	-0.05	0.63	-0.08	0.934
Compexp/Compexp	-0.01	0.82	-.0.02	0.985
LOC/Intercept	-0.47	0.91	-0.52	0.603
LOC/Gender	0.27	0.61	0.45	0.650
LOC/Npgrmcs	-0.14	0.58	-0.24	0.814
LOC/owncomputer	0.73	0.63	1.13	0.258
LOC/Mbcguni	-0.09	0.58	-0.15	0.883
LOC/Cppbgd	-0.48	0.63	-0.76	0.445
LOC/Compexp	0.01	0.58	0.01	0.992
LOC/LOC	0.00	0.82	0.01	0.996

LEVEL 1	TAU-HAT	STD.ERR.	Z-VALUE	PR > |Z|
intcept /intcept	226.77	0.09	2436.20	0.001

We estimate the student-level (intrinsic factors) variance in Java programming Self efficacy byusing equation 4.1

$\frac{\Phi_1}{\Phi_2+\Phi_1}$ = Total Variance ....(4.1)

Where $\Phi_2$represent between group variability (Variance of level-2) and $\Phi_1$represent within group variability (variance of level-1).

From table 4.8, Φ_2 = 2.15 and Φ_1 = 226.77

:. Total variance in Programming self efficacy = $\frac{\Phi_1}{\Phi_2+\Phi_1}$......................................(4.2)

This is given as $\frac{226.77}{2.15+226.77} = \frac{226.77}{228.92}$ = 0.99

Therefore the total variance in Java programming self efficacy accounted for bystudent-leveldifferences is 0.99. This result indicates that about 99.0% of the student-level variation in programming self efficacy is explained by the differences in undergraduates' intrinsic factors. However, it is the interaction of computer ownership with gender, number of programming courses, mathematics background and background in C++; C++ with gender, number of programming courses, and mathematics background that accounted for the observed variance. The other individual explanatory factors such as computer experience, and locus of control as well as their interactions do not significantly contribute to the observed variance in computer self efficacy.

Discussion

The result from table 4.7 which is the fixed part of the Linear Growth Model 1 of Intrinsic factorsshowed that only number of programming courses taken before Java class contributed positively to variation in Java programming self efficacy. The result also showed that 99.0% of the variance in Java Programming Self Efficacy across the institutions was accounted for jointly by student-related factors (computer experiences, LOC, gender, mathematics background in C++ and number of programming courses offered before entering the Java class). The implication is that students' background variables are forces to reckon with in the effort to improve Java Programming Self Efficacy of Computer undergraduates in our various institutions.

Computer experience in this study did notcontribute significantly to the variation in Java programming self efficacy. This finding agrees with that of Jegede (2009a) in a study carried out among Engineering students in a South west Nigerian University. The finding however contradicts that of Askar &Davenport (2009).In their study of factorsrelated to Java programming self efficacy among engineering students in Turkey they found that the number of years of computer experience had a significant linear contribution to Java programming self-efficacy scores. Since the findings of this study agrees with the one of Jegede (2009) conducted within the same region of Nigeria but contradicts that of Askar and Davenport (2009) carried out in Turkey, it is reasonable to think that the location and the level of technological advancement might be having a stronger influence on programming self efficacy than computer experience is having.

Research Question Six: How much of the student-level variance in Java programming achievement of computer undergraduates is associated with gender, ownership of computer, mathematics background, background in C++, computer experience LOC and number of programming courses taken before entering Java class.

To answer this research question a multilevel analysis was conducted with ordinary least square option of LISREL. The model used is known as model 1in that explanatory variables of studen-level differences plus the intercept of institution-level (macro level) were entered. That is the focus was on linear growth model with random intercepts and their slopes

Statistically, the growth modelling for variance decomposition of achievement is given by

achievement$_{ij}$ = β_0 +β_1 X Gd$_{ij}$ +u$_{0i}$ +u$_{ij}$ XGd$_{ij}$+ .... +β_n X NPCS$_{ij}$ +u$_{0i}$ +u$_{ij}$ XNPCS$_{ij}$+ e$_{ij}$

Where achievement$_{ij}$ represents score "j" for student "i"; β_0 represents the intercept of the fixed part of the model andu$_{oi}$ represents the random variation in intercepts at level – 2 of the model and eij denotes the random variation at level – 1 of the model,u$_{ij}$ represents the random variation in slopes for each of the intrinsic factors.Table 4.9and 4.10 present the fixed part of the model and random part of the model.

Table4.9: Fixed Part of the Linear Growth Model 1 of Intrinsic factors

COEFFICIENTS	BETA-HAT	STD.ERR	Z-VALUE	PR > |Z|
Intcept	7.26	9.68	0.75	0.451
Gender	2.75	2.43	1.130.261	
Owncomp	2.33	3.02	0.77	0.440
MBCGUNI	0.65	0.49	1.31	0.191
CPPBGD	3.74	2.44	1.53	0.126
NPGRMCS	1.21	0.65	1.84	0.065
COMPEXP	-0.020.08	-0.29	0.772	
LOC	-0.03	0.05	-0.54 0.587	

DEVIANCE= -2*LOG(LIKELIHOOD) = 64192.09
NUMBER OF FREE PARAMETERS = 45

Table 4.9 shows that none of the seven intrinsic factors considered, significantly contributed to the prediction model that is undergraduates programming achievement.

Table 4.10: Random Part of the Linear growth model 1 of Intrinsic Factors

LEVEL 2	TAU-HAT	STD-ERR	Z-VALUE	PR >|Z|
Intercept/Intercept	-82.65	1.83	-45.16	0.001
Gender/Intercept	23.30	0.94	24.71	0.001
Gender/Gender	-16.64	0.90	-18.55	0.001
Npgrmcs/Intercept	17.03	0.97	-17.60	0.001
Npgrmcs/Gender	18.88	0.67	27.99	0.001
Npgrmcs/ Npgrmcs	-8.67	1.00	-8.67	0.001
Owncomputer/Intercept	6.23	0.91	6.84	0.001
Owncomputer/Gender	-0.85	0.61	-1.40	0.162
Owncomputer/Npgrmcs	-2.34	0.65	-3.61	0.001
Owncomputer/Owncomputer	-0.91	0.82	-1.10	0.270
Mbcguni/Intercept	73.86	0.99	74.42	0.001
Mbcguni/Gender	-9.96	0.66	-15.08	0.001
Mbcguni/Npgrmcs	-19.65	0.71	-27.77	0.001
Mbcguni/owncomputer	5.32	0.63	8.40	0.001
Mbcguni/ Mbcguni	-17.95	0.96	-18.61	0.001
Cppbgd/Intercept	14.81	0.91	16.21	0.001
Cppbgd/Gender	0.35	0.61	0.58	0.563
Cppbgd/Npgrmcs	-2.31	0.65	-3.28	0.001
Cppbgd/owncomputer	1.01	0.58	1.74	0.082
Cppbgd/Mbcguni	5.64	0.63	8.91	0.001
Cppbgd/Cppbgd	-0.68	0.82	-0.83	0.408
Compexp/Intercept	-0.17	0.91	-0.18	0.855
Compexp/Gender	-0.34	0.61	-0.57	0.569
Compexp/Npgrmcs	0.21	0.65.	0.32	0.747
Compexp/owncomputer	-0.04	0.58	-0.08	0.938
Compexp/Mbcguni	-0.24	0.63	-0.39	0.699
Compexp/Cppbgd	-0.13	0.58	-0.23	0.822
Compexp/Compexp	-0.01	0.82	-0.01	0.990
LOC/Intercept	-0.54	0.91	-0.59	0.553
LOC/Gender	0.40	0.61	0.66	0.509
LOC/Npgrmcs	0.73	0.65	1.13	0.257
LOC/owncomputer	-0.09	0.58	-0.16	0.877
LOC/Mbcguni	-0.49	0.63	-0.78	0.434
LOC/Cppbgd	-0.15	0.58	-0.26	0.794
LOC/Compexp	0.01	0.58	0.02	0.958
LOC/LOC	0.00	0.82	0.00	0.998

```
-----LEVEL 1              TAU-HAT     STD.ERR.     Z-VALUE   PR > |Z|
------------------------------------------------------------------------------
Intcept /Intcept          273.720.092940.54   0.001
```

We estimate the student-level (intrinsic factors) variance in Java programming achievement by using equation 4.1

$\frac{\Phi_1}{\Phi_2+\Phi_1}$ = Total Variance ....(4.1)

Where $\Phi_2$represent between group variability (Variance of level-2) and $\Phi_1$represent within group variability (variance of level-1).

From table 4.10, Φ_2 = 182.60 and Φ_1 = 273.72

:. Total variance in Programming achievement = $\frac{\Phi_1}{\Phi_2+\Phi_1}$......................................(4.2)

This is given as $\frac{273.60}{182.60+273.72} = \frac{273.72}{456.32}$ = 0.60

Therefore the total variance in Java programming achievement accounted for bystudent-leveldifferences is 0.60. This result indicates that about 60.0% of the student-level variation in programming achievement is explained by the differences in undergraduates' intrinsic factors. However, it is the interaction of computer ownership with gender, number of programming courses, and background in C++; background in C++with gender, number of programming courses, and mathematics background accounted for the observed variance. The other individual explanatory factors such as computer experience, and locus of control as well as their interactions do not significantly contribute to the observed variance in computer programming achievement.

Discussion

The result from table 4.9 which is the fixed part of the Linear Growth Model 1 of Intrinsic factors showed that none of the intrinsic factors contributed significantly to the variance in programming achievement. The result from table 4.10 however showed that the interaction of computer ownership with gender, number of programming courses and background in C++; background in C++ with gender, number of programming courses and mathematics background accounted for the observed variance. The finding of this study contradicts the findings from the studies of Byrne and Lyons (2001), Hagan and Markham (2000); Ramalingan, Labelle and Wiedenbeck (2004); Wilson and Shrock (2002). In their various studies with different categories of participants, they all found out that previous programming experience had a positive effect in introductory programming courses.The contradiction between the findings of this work and the previous works above might be because they all used students taken introductory courses in computer programming which is not the case with the present study.

The result from table 4.8 showed that the combination of differences in students' gender, LOC, mathematics background, computer ownership, background in C++, number of programming courses taken before entering Java class and computer experiences contributed about 60% to the total variation in programming achievement across the institutions.

The positive contribution of computer experience corroborates the findings of Byrne and Lyons (2001), Hagan and Markham (2000); Ramalingan, Labelle and Wiedenbeck (2004); Wilson and Shrock (2002). In their various studies with different categories of participants, they all found out that previous programming experience had a positive effect in introductory programming courses. It appears that there is a consistency in the finding that previous programming experience impacts positively on performance of students at various levels in programming examination.

LOC did not contribute positively to programming achievement. This result agrees with the findings of Jegede (2009b). Jegede (2009b) found that programming achievement had no significant relationship with the faith students have in their own lives (internality), the belief in the irresistible power of others on their lives (powerful others) and the trust they place on chance in determining their course in life (chance).However, Bishop-Clark (1995) and Chim and Zecker (1985) opined that LOC could be used to explain variability in programming achievement. Obviously, the claims of Bishop-Clark (1995) and Chin and Zecker (1985) were opposed to that of Jegede (2009b) and this study.

This may be because of the following two reasons: (i). the Bishop-Clark (1995) and Chin and Zecker (1985) merely expressed their opinions, their claims were not research findings but mere opinions. This study and that of Jegede (2009b) were products of research findings; (ii). Bishop-Clark (1995) and Chin and Zecker (1985) made their claims 17 years and 27 years ago respectively. Jegede's (2009b) study and the present study are more recent. Given these two reasons, the opposing views and claims are expected. Conclusively, LOC does not seem to impact significantly on programming achievement.

The findings of this study showed that mathematics background did not contributesignificantly to programming achievement. There is a dearth of literature on the influence of mathematics background and performance in programming. There are however researches on the influence of mathematics ability on programming achievement. The finding of this study is opposed to that of Wilson and Shrock (2002) and Byrme and Lyons (2001). The two studies showed a positive relationship of performance in mathematics and computer programming. Interestingly, there had been a kind of consensus that mathematics ability predicts performance in computer programming. Erdagan, Aydin and Kabaca (2007) submits that computing as a subject requires a structure and approach with which students have some experience and similar cognitive skill used in the study of mathematics. Byrne and Lyons (2001) also opined that mathematics aptitude should be a pre-requisite for acceptance into computer science programs. There is therefore a contradiction between the findings of this study on the influence of mathematics background on computer programming achievement and the previous studies on the influence of mathematics ability on programming achievement. Mathematics background in this work was measured by the number of mathematics courses the respondents had taken as at the time of the study. This implies that the more mathematics courses taken the lower the performance in Java programming.This trend might be explained by the fact that the types of mathematics courses taught are not relevant to programming There is therefore the need to ensure that the curriculum is revisited so that courses that are pre-requisites for good performance in computer programming are available to the computer undergraduates and also properly handled by experienced instructors.

The result also showed that 60% of the variance in programming achievement was accounted for by student level differences. This is in agreement with the findings of Vanden Broeck, Van Damme and Opdenakker (2005). The authors analysed the effects of student-teacher – and school-level factors on students' achievement in Belgium. Two classes from each school were selected; this made it possible to build three-level hierarchical model. As a national option, the extended versions of student, teacher and school questionnaires were used. By means of null model, it was found that almost 58% of the total variance in mathematics scores was situated at the student level, 28% was due to differences between classes and 14% was due to differences between schools. Infact, after adding student level factors to the model, some of the class and school level factors were no longer significant. The finding of this study which was also corroborated by Van den Broeck, Van Damme and Opdenakker (2005) has implication. The implication is that students' background variables are also forces to reckon with in the effort to improve the general performance of Computer undergraduates in Computer programming in our various institutions.

HypothesisOne: There is no significant difference, in the mean score of self-efficacy in Java programming, between undergraduates in Federal and State Universities

To test the hypothesis, independent smple t-test was used.

Table 4.11: T-test comparison of Self-Efficacy in Java Programing, BetweenUndergraduates in Federal and State Universities

Inst. Type	**N**	**Mean**	**S.D**	**t_{cal}**	**Df**	**p-value**	**Remark**
Federal	194	128.05	44.57	7.57	252	0.001*	S
State	60	173.97	26.39				

S – Significant

Table 4.11 presents the t-test comparison of the scores ofself-efficacy in Java Programming, between undergraduates in Federal and state universities. The t-test comparison showed a statistically significant difference between the mean scores of self-efficacy in Java Programming, between undergraduates in Federal and State Universities(Tcalculated = 7.57, df = 252, p < 0.05). We therefore reject the null hypothesis.Therefore there is a significant difference in the mean score of self-efficacy in Java programming, between undergraduates in Federal and State Universities.

Discussion

The result in table 4.11 showed a significant difference in the mean self-efficacy scores across institution type; with the state Universities having higher mean scores. This finding is in agreement with the finding of Gisemba (2011) in a study carried out to determine the extent to which teacher self efficacyand students mathematics self efficacy could enhance secondary school students' achievement. Teachers form part of the institution and hence coud be used to explain institutional differences. In the study by Gisemba (2011), it was found that teachers frequent use of mathematics homework and level of interest and enjoyment of mathematics as well as their ability and competence in teaching mathematics played a key role in promoting students mathematics self efficacy.To boost students' self efficacy in computer programming, students should not only be the focus, the institutions must also create an enabling environment. In this work the mean scoresin table 4.11 showed a higher mean self-efficacy score from the undergraduates in state university. It therefore follows that the meanself-efficacy in Java Programming of undergraduates in the state university(mean = 173.97, standard deviation = 26.39) is significantly higher than that oftheir counterparts in the Federal universities(mean = 128.05; standard deviation = 44.57). The mean self-efficacy score in the state-owned institution could be explained by the fact that unlike in the federal Universities, the state Universities are used to providing and doing things for themselves. Consequently ownership of computer which is expected to be more in the state owned Universities might have affected their self-efficacy in Java programming.

Hypothesis Two: There is no significant difference, in the mean score of achievement in Java programing, between undergraduates in Federal and State Universities.

Table 4.12: T-test comparison of Achievement in Java Programing, Between Undergraduates in Federal and State Universities.

Inst. Type	N	Mean	S.D	t_{cal}	df	p-value	Remark
Federal	194	20.54	18.72	8.67	252	0.250	N.S
State	60	22.92	11.78				

N.S – Not Significant

Table 4.12 presents the t-test comparison of the scores ofachievement in Java Programing, between undergraduates in Federal and state universities. The t-test comparison showed a difference which is not statistically significant between the mean scores of achievement in Java Programing, between undergraduates in Federal and State Universities(Tcalculated = 8.67, df = 252, p > 0.05). We therefore accept the null hypothesis.Therefore there is no significant difference, in the mean score of achievement in Java programing, between undergraduates in Federal and State Universities.

Discussion

The result in table 4.12 showed a difference which is not statistically significant in the mean achievement scores across institution type. Variance in achievement due to institutional – differences varies from one place to another. Park and Park (2006) found that in SouthKorea, about 4% of the total variance of mathematics achievement was due to institutional – level factors. For south african students it was 55% (Howie, 2006). Similarly for Australian students 27% and 47% formed the percentage contributions of institutional – level differences in the Trends in mathematics and science study (TIMSS) conducted in 1995 and 1999 respectively (Fullarton, 2004). In Singapore 45%, Botswana 27%, Chiki 30% and Flenders 14% were the percentage contributions found (Chepete, 2008, Mohammadpour, Maradi & Najid, 2009; Ramirez, 2006; VandenBrock, VanDamme, Opdenkker, 2006). Recently, a study carried out by Ghagar, Othman, MohammadPour (2011) among eight graders in Malaysia indicated that 57.28% of the total variance in mathematics achievement of eight graders in Malaysia was accounted for by institutional level differences. Also Stemler (2001) studied school effectiveness in mathematics and science at the 4th grade, using data from International Educational Assessment (IEA's) TIMSS study. 14 countries were included in the study. In general, about one – quarter (25%) of the variability in mathematics and science achievement was found to lie between schools. In the present study however, state University had higher mean achievement scores when compared with their federal University counterparts; although the difference is not significant. The difference in the mean achievement in Java programming across institution type as observed among the respondents of this study might also be explained by the fact that the state Universities are used to providing and doing things for themselves. Consequently ownership of computer which is expected to be more in the state owned Universities might have affected their achievement in Java programming.

CHAPTER FIVE

SUMMARY OF FINDINGS, IMPLICATIONS, RECOMMENDATIONS AND CONCLUSION

This chapter presents the summary of findings discussed in chapter four and the educational implications of the study. The recommendations, limitations of the study as well as suggestions for further research are also presented .

5.1 Summary of Findings

Observation and research have shown that learning computer programmingis a problem to many computer majors. By design, the computer curricula in various Universities are supposed to produce graduates with positive self efficacy and skill in writing programs some of which are also to take to software development as their future career. Due to the difficulty encountered in learning programs at school, many of them graduate neither having the required competence in programming nor even having confidence in their ability to program (self efficacy). Consequently many of them do not take to programming and software development after graduation.

Ironically, programmers are now much in demand in the information Technology industry because of the increasing need of application software in the operations of medium and large scale industries, schools, government establishments etc . Acquisition of skills and self efficacy do not exist in isolation, they are influenced by some factors. Some of those factors are peculiar to the students (intrinsic). The institution factor (extrinsic) is also germane. In this survey study, intrinsic factors (computer experience, gender, mathematics background, computer ownership, Locus of control, background in C++ and number of programming courses offered before entering Java class) were measured and their relationship with computer undergraduates' achievement and self efficacy in Java programming language was investigated.

Because of the hierarchical nature of the data collected (the computeer undergraduates studied are nested within their various institutions), a multilevel analysis study of the data was done to ascertain the variation in dependent variables (Achievement and self efficacy in Java programming) caused by the interaction of both the student related factors (intrinsic factors) and their institutions (extrinsic factors).

The major findings of this study are summarised below:

i. The Java programming self efficacy scores of the computer undergraduates was found to be above average. Achievement in Java programming was however found to be below average.

ii. The computer undergraduates took more of mathematics courses than programming courses. Mathematics background related positively and significantly with Java programming self efficacy. The same mathematics background correlated negatively with Java programming achievement.

iii. Mathematics background, computer experience, Locus of control, Background in C++ and Number of programming courses taken before entering the Java programming class all had positive and significant relationships with Java programming self efficacy.

iv. Number of programming courses taken before entering the Java programming class and Java programming self efficacy both had a positive significant relationship with Java programming achievement of computer undergraduates.

v. 91.0 percent of the total variance in programming self efficacy of the computer undergraduates was accounted for by institution level differences. The remaining 9.0 percent was accounted for by the student – level differences.

vi. 82.0 percent of the total variance in Java programming achievement of the computer undergraduates was explained in the study by the differences in their institutions. The remaining 18.0 percent was accounted for by the student-level differences.

vii. The fixed part of the Linear growth Model 1 of Intrinsic factors showed only the number of programming courses offered before entering Java programming class contributed significantly to the variation in computer undergraduates' Java programming self efficacy.

viii. The combination of differences in students'computer experiences, gender, LOC, mathematics background, computer ownership, background in C++ and number of programming courses taken before entering Java programming class contributed about 99% to the total variation in programming self efficacy.

ix. The fixed part of the Linear growth Model 1 of Intrinsic factorsshowed that none of the factors contributed significantly to the variance in Java programming achievement of the computer undergraduates.

x. 60.0 percent of the variance in Java programming achievement were accounted for jointly by the intrinsic factors (computer experiences, LOC, gender, mathematics background, background in C++ and number of programming courses before entering the Java class).

5.2 Implications And Recommendations

The findings of this study (on the multilevel analysis of intrinsic and extrinsic factors predicting self efficacy and achievement in computer programming of computer undergraduates in South – West Nigeria) have a number of useful implications and recommendation for stakeholders in education and IT industries.

(i) The level of computer undergraduates'confidence in their ability to write programs using Java (self efficacy) was far higher than the competence displayed by them in the Java programming achievement test (Achievement in Java programming test). Obviously, the computer undergraduates were not exposed to enough exercises in programming. Since practice make for perfection, it is therefore recommended that students are given enough experiences to practice with, individually and in groups. Tutorial classes on programming that will afford the students the opportunity to interact freely with their instructors should also be organised in each of the programming courses at the departmental level. The chapters of National Association of Computer Science Students (NACOSS) in each university should also corroborate the efforts of the department in this regard. The Nigeria computer society (NCS), the umbrella association of computer professionals should organise seminars, workshops as well as competitions on programming principles and practice. More programming courses should also be introduced at the secondary school level.

(ii) The computer undergraduates that took more programming courses did better in Java programming achievement. Also those with higher Java programming self effciacy did better in Java programming achievement compared with their counterparrs with lower Java programming self efficacy. Since programming forms the heart of computing and there is a dearth of programmers, computer undergraduates should be exposed to more programming courses. Attempts should be made to increase computer undergraduates'programming self efficacy using the knowledge of the four sources of self efficacy (mastery experiences, vicarious experiences, verbal persuasion and psychological traits). For instance, since mastery experiences could boost self efficacy, the test questions given at the initial stage should be relatively easy for the beginning programmer. Also they should not be tested on what they have not been taught. If these are put in place, they are likely to perform well in those inital tests. Good performance in the initial tests would in turn build a robust sense of self efficacy in them which would in turn increase their performance in programming. Opportunities should also be given to them to watch their colleagues write programs. Seeing this will enhance their self-efficacy that they could do it. This is what is called Vicarious Experiences in self-efficacy theory. Besides, Verbal Persuasion in the class by their lecturers according to self-efficacy theory would boost their sense of self-efficacy. Beyond facilitating in the programming class, the lecturers also need to be trained to also ensure that the students are in physical and emotional state that will enhance their self-efficacy in Java programming. In summary, the Java programming lecturers need to be trained on how to take advantage of the four sources of self-efficacy to enhance their students' self-efficacy in the programming language of instruction. If the knowledge of sources of self-efficacy is properly applied in the class; it is hoped that students' sense of self-efficacy would be enhanced and consequently achievement would be positively affected.

(iii) Higher scores in the measurement of five of the student related variables (mathematics background, computer experience, LOC, background in C++ and the number of programming courses taken before entering the Java programming class) related with higher scores in Java programming self efficacy. The implication of this findings is that, to increase the level of confidence students have in their ability to program using Java, the computer undergraduates should be made to take more mathematics and programming (C++ language inclusive) courses before the Java programming class. They should also be exposed to more computer algebra softwares instead of packages.

(iv) Mathematics background (signified by the number of mathematics courses taken by the students) correlated positively and significantly with Java programming self efficacy. The relationship between mathematics background and Java programming achievement is however very low and also not significant at $p < 0.05$.The implication is that the more mathematics courses they took, the higher their self efficacy in Java programming. However, the number of mathematics courses they took did not significantly relate with their achievement in Java programming. This research findings especially on the relationship of mathematics background and Java programming achievement is in contrast with many of the findings on the relationship of mathematics achievement and Java programming achievement. Therefore, it is reasonable to conclude that taken many mathematics courses without the necessary skills acquired may not significantly impact on achievement in a mathematics related area like computer programming. It is therefore recommended that some Computer Integrated Mathematics courses where some computer topics are introduced into the curricula of those mathematics courses should be introduced. Those courses could be given codes in the computer departments and taught by computer lecturers.

(v) The joint contribution of the student-level factors (computer experiences, gender, mathematics background, computer ownership, LOC, background in C++ and number of programming courses taken before entering the Java programming class) was 9.0 percent. However the institution type contributed 91.0 percent to the variation in computer undergraduates self efficacy in Java programming. The implication is that the contributions of institution-level (Level -2) variables far outweigh that of the student-level (Level - 1). Therefore creating a good environment for learning programming is germane.A conducive and enabling environment for effective learning and acquisition of programming skills must not be neglected. Compilers of various computer languages as well as other materials useful for learning programs should be installed in the computers that are in the computer laboratories. Pair programming strategy should be employed in the during instructions. Discussion groups should also be encouraged among the computer undergraduates. The quality assurance unit of the Universities must also monitor students'attitude to learning and variables peculiar to them that could increase or decrease achievement in programming.

(vi). As high as 60.0 percent of the variation in Java programming achievement across the institutions were accounted for jointly by student – level (Level -1) variables.

Studentsshould be part of the focus in intervention measures to be put in place in order to increase students' self efficacy in programming. The notion that they are matured enough and should be taught anyhow should be discarded.Beyond covering the course content, it is recommended that lectures and instructors are mindful of these student-level variables (Level -1) in this study. The findings of this study has shown that non-cognitivevariables contribute meaningfully to Java programming achievement and self efficacy of computer undergraduates. This finding therefore confirms the need for professionalism in teaching at all levels of education (Universities inclusive). In order to stop producing graduates that are neither competent nor confident in their ability to do what they are trained for, those lecturing in the universities (at least in computer) should in addition to the Knowledge of what they are teaching and the skills required to teach, be made to go through some trainings in human psychology and necessary pedeagogiacl strategies.

(vii) The combination of differences in the intrinsic factors (computer experiences, gender, mathematics background, computer ownership, LOC, background in C++ and number of programming courses offered before entering Java programming class) contributed about 99 percent and 60.0 percent respectively to the variations in self efficacy and achievement. The implication is that the factors peculiar to students together have very high influence on the self efficacy and performance in Java programming. Again, beyond creating an environment conducive for learning and monitoring how lecturers facilitate in their classes. The authorities of the institutions would also need to provide counselling services for those computer undergraduates on the personality factors in this study. The information provided in the findings of this study should also be used regularly by course advisors in guiding their students on choice of courses and their experiences on the programme. Counselling units could design relevant instruments for them to fill. Based on the data gathered and the subsequent findings, workshops, group and/or individual counselling sessions could be organised for the students to address the challenges discovered.

5.3 Conclusion

This study utilised the Multiple Linear Regression and Multilevel analysis using the Statistical Package for Social Sciences (SPSS) and Linear Structural Relations (LISREL) software packages respectively. The influence of some (student level variables) and institution-level variable on both the Java programming self-efficacy and achievement of computer undergraduates in Public Universities in South – West, Nigeria was examined.

The institution-level variable was seen to have some influence on students' self efficacy and achievement in Java programming (91.0% and 82.0% respectively) of computer undergraduates. The combination of student level variables (computer experiences, gender, LOC, mathematics background, computer ownership, background in C++ and number of programming courses taken before entering Java programming class) contributed as high as 99.0% and 60.0% respectively to the variation in Java programming self efficacy and Java programming achievement across the various institutions.

It is important that the computer undergraduates themselves, computer lecturers, the university authorities and the monitoring agencies (such as the Nigeria Universities Commission, NUC as well as the Federal and State ministries of education) utilise the research findings to improve Java programming self efficacy and achievement in computer programming. All the stakeholders highlighted above should therefore think seriously about implementing the various suggestions proposed in this work. By so doing, to a great extent the difficulties encountered in learning programming and low self efficacy may be overcome. This would in turn help the various computer departments in our Universities to produce graduates who are not only competent but have confidence in their ability to program. This would in turn solve the problem of the dearth of programmers and software experts in our IT industry. This, if done would eventually bring about the realisation of that vision of a Nigeria which is IT capable and self reliant andalso a key player in the information society.

5.4 Limitations And Suggestions For Further Studies.

This study was limited to the South Western part of Nigeria. It was also limited to Federal and State owned institutions. There is the need for a replication of the study in the other geo – political zones of the country. A more elaborate study which will cover all the Universities in the entire country could be undertaken. An attempt should also be made to include private institutions in the study. This would afford the comparison across institution types.

The multilevel analysis was limited to two – levels in this study. A more elaborate multilevel research at three or more levels could also be done. For instance student – level, classroom – level, institution – level variables could be considered simultaneously.

The findings of this study borders on computer majors in Public Universities in South – West, Nigeria. It is suggested that similar studies be carried out in Polythecnics, Colleges of Education and even Secondary schools. The study could also be carried out among computer non – majors in the various tertiary institutions.

An experimental study on instructional strategies in computer programming aimed at recommending the best instructional approach that could produce graduates that would be interested in and competent to program will provide useful result.

Finally, this type of research should be an ongoing one repeated from time to time to see whether the situation has improved.

REFERENCES

Adams, C.M., & Forsyth, P. B. 2006. Proximate sources of collective teacher effciacy. *Journal of Educational Administration*, 15, pp.625 - 642

Agarwal, R., Sambamurthy, V., and Stair, R. 2000.Research Report: The Evolving Relationship between General and Specific Computer Self-Efficacy-An Empirical Assessment.*Information Systems Research*, 11(4) pp. 418-430.

Aitkin M, Anderson D and Hinde, J. 1981. Statistical modelling of data on teaching styles (with discussion). *Journal of the Royal Statistical Society A* (144), pp. 148 – 161

Akyuz, G., 2006.Teacher and Classroom characteristics: Their Relationship with Mathematics Achievement in Turkey, European Union Countries and candidate countries. A thesis submitted to the Graduate school of Natural and Applied Science of Middle East Technical University in partial fulfillment of the requirements for the doctor of Philosophy in secondary science and mathematics Education.

Amadi, M. 2010. Affective determinants of ESL success. Unpublished M.Ed project, department of teacher education, University of Ibadan.

Aptech Limited 2005. Java Simplified -1. Aptech Limited. Mumbai.

Araromi, M 2010. Motivation, Verbal Ability, Attitude, Gender And Locus Of Control As Predictors Of Academic Achievement In French. A seminar paper. In Fakeye D.O (2011). Locus of Control as a correlate of Achievement in English Language as a second language in Ibadan. The Journal of International Social Research; 4 (17), pp 546-552.

Askar, P. & Davenport, D. 2009. An investigation of factors related to self-efficacy for Java Programming Among Engineering Students. *Turkish Online Journal of Education Technology 8(1)*

Babbage, C. 1961.*Charles Babbage and his Calculating Engines.*In Morrison P.And Marrison E. (Eds). New York: Dover.

Bandura, A. 1977a. Self efficacy: Towards a Unifying Theory of Behavioural Change. Psychological Review. 27(1), 13-15

_________1977b. Self Efficacy: Towards a Unifying Theory of behaviour. Psychological Review, 84(2), 191-215. Retrieved from the World Wide web:http//www.ncbi.nih.gov/pubmed/847064.

_________1986.*Social foundation of thought and actions: A social cognitive theory.* Englewood cliffs, New Jersey: Prentice Hall.

_________ 1993. Perceived Self Efficacy in Cognitive Development and Functioning. *Educational Psychology*. 28, 117-148

__________1994a. Perceived Self Efficacy in Cognitive Development and Functioning (Electronic Version).*Educational Psychologist* 28(2), 117 – 148

__________1994b. Self eficacy.In V.S Ramachaudran (Ed.), Encyclopedia of human behaviour, 4, 71 – 81. New York: Academic Press. Retrieved from the World Wide Web, http://des.emory.edu/mfp/BanEncy.html

__________ 1996. *Assessing self-efficacy beliefs and academic outcome: The case for specific city and correspondence.* A paper presented at the annual meeting of the American Educational Research Association, New York, NY.

_________1997. Self – Efficacy: The exercise of self control. New Terk; W.H Freeman and company.In Shaw N.E 2008.The Relationship between Perceived Parenting style, Academic Self-Efficacy and College Adjustment of Freshman Engineering Students.Master of Thesis. University of North Texas.

__________2006a. Adolescent Development from an Agentic Perspective.In F. Pajares and T. Urdan (Eds.).Self Efficacy beliefs of Adolescents (pp. 1-43). Greenwich, Connecticut: Information Age publishing.

__________ 2006b. Guide for Constructing Self Efficacy Scales. In F. Pajares & T. Urdan (Eds.).Self Efficacy beliefs of Adolescents (pp. 307-337). Greenwich Connecticut: Information Age publishing.

Beas, M.I & Salanova, M. 2006.Self-efficacy Beliefs, Computer Training and Psychological Well-Being among Information and Communication Technology Workers.*Computers in Human Behavior, 22*, 10431058. Impact Index JCR = 0.808

Begum, M. 2003.An Ontology for Teaching Programming. *Association For Computer Machinery* 43, 69 – 182

Bergin, Thomas J. And Richard G. Gibson, (Eds) 1996.History of programming Languages-II. New York: ACM Press.

Berkakatin (1995). Objects-oriented programming in C++ PHI 1995.

Berndt, T. J., & Keefe, K. 1992. Friends' influence on adolescents' perceptions
BETA programming Language, Addison – Wesly/ ACM Press, 1993.

Bennedsen, J. And Caspersen M.E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? SIGCSE Bull., 3812; 39-43 1138430.

Bishop-Clark, C. 1995.Cognitive Style Personality and Computer Programming.*Computers in Human Behavior*, 11, 241-260

Bolan, S. (2000). Women in IT on decline .Computing Canada, 26 (22), 29

Bozionelos, N. 2001. Computer Experience: Relationship with Computer Experience and Prevalence. *Computers in Human Behaviour,* 17, 213-224

Busch, T. 1995. Gender Differences in Self Efficacy and Attitudes towards Computers.*Journal of Educational Computing Research*, 12, 147-158.

Byrne, P., & Lyons, G. 2001. The Effect of Student Attributes on Success in Programming. ITiCSE: *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education.* ACM Press, NY, 49-52

Carter ,P. 1997. An introduction to the Java programming language www.cs.binghamton.edu/~guydosh/cs350/Javaprimer.pdf last accessed, November 7,2012, 5:05pm

Cassidy, S and Eachus, P. 2002. Developing the Computer User Self-Efficacy (CSUE) Scale: Investigating the Relationship between Computer Self-Efficacy, Gender and Experience with Computers. *Educational Computing Research*, 26(2), 133-153.

Chepete, P. 2008. Modeling of the Factors Affecting Mathematics Achievement of Form 1 Students In Bostwana Based On The 2003 Trends In International Mathematics And Science Study. Unpublished doctor of philosophy, Indiana University.Child Quarterly, 42, 39-48.

Chilson, I. M, Carey, J., and Hernandez. 2002. Information Technology Skills for a Pluralistic Society: Is the playing field level? *Journal of Research on Technology in Education* 35: 38 – 79.

Chin, J.B and Zecker, S.G. 1985. Personality and Cognitive Factors Influencing Computer Programming Performance. Paper Presented at the Annual Meeting of the Eastern Psychological Association, Boston. March 21-24, 1985.

Chung, C. 1988. Correlates of problem solving in programming.*The Chinese University Educational Journal*, 16(2), 185 – 190.

Clegg, S. Trayhura, D. (1999). Gender and computing: not the same old story, *British Educational Research Journal*, 25, 5 pp 75-89

Cohoon J.M (2001). Towards improving female retention in the computer science major.Communications of the ACM 44(5); 108-114.

Compeau, D.R. & Higgins, C.A. 1995. Computer Self-Efficacy: Development of a Measure and Initial Test. *MIS Quarterly*, 19 (2), 189-211*Computer Studies* 51, 71- 87

Czaja, S.J., Charness, N., Fisk, A.D., Hertzog, C., Nair, S.N., Rogers, W.A, et al 2006. Factors Predicting the use of Technology: Findings from the Center for Research and Education on Aging and Technology Enhancement. *Psychology and Aging* 21(2), 333-352

Debacker, T.K and Nelson, R.M (2000). Motivation to learn science: Differences related to gender, classtype and ability. Journal of Education Research, 93(4), 245-255.

Downey, J.P., & McMurtrey, M. (2009). Introducing task – based general computer self-efficacy: An empirical comparison of three general self-efficacy instruments. Interacting with computers19, 382-396

Doyle E., Stamouli, I., & Huggard. M. 2005. Computer Anxiety, Self – efficacy, Computer Experience: An Investigation through a Computer Science Degree. A Paper Presented at the 35th ASEE / IEEE Frontiers in Education Conference, Indiana Polis, IN.

Durndell, A., Hagg, Z., & Laithwaite, H. 2000. Computer self-efficacy and gender: A cross Cultural study of Scotland and Romania.*Personality and Individual Differences.* 28, 1037 – 1044

Emeke E.A., Adeoye, H.A., & Torubeli, V.A. 2006. Locus of Control, Self Concept and Emotional Intelligence as Correlates of Academic Achievement among Adolescents in Senior Secondary Schools in Oyo State. *Nigerian Journal of Clinical and Counselling Psychology 12 (2)*, 122-137.

Emeke E.A., & Yoloye, T.W., 2000. An Investigation into the effect of Locus of Control on the Adjustment of Foreign Students at University of Ibadan, Ibadan, Nigeria. *African Journal of Cross Cultural Psychology and Sport Facilitation* Volume 2 pp 31-35.

Erdogan, Y., Aydin, E., Kabaca, T. 2007. Exploring the psycholgical predictors of programming achievement, *Journal of Instructional Psychology*, September. 2008

Ertmer, P.A., Evenbech, E., Cennamo, K.S., & Lehman, J.D. 1994. Enhancing self-efficacy for computer technologies through the use of positive classroom experience 42(3), 45 – 62.

Fakeye D.O. 2011.Locus of Control as a Correlate of Achievement in English as a Second Language in Ibadan.*The Journal of International Research* 4(17), 546 – 552.

Fergusson A.D. 2000.A history of computer programming Languages.www.csbrown.edu/~adf/prgramming_languages.html. Last Accessed:November 7,2012, 5:05pm.

Fowler, L., Campbell, V., McGill, D. & Roy G. 2002. An Innovative Approach to Teaching First year programming supported by learning style investigation. Paper presented at the Australasian Association for Engineering Education, Melbourne.

Fullarton, S. 2004. Closing the gaps between schools, accounting for variation in mathematics achievement in Australian schools using TIMSS 95 and TIMSS 1999. Paper presented at the The 1st IEA International Research Conference, IRC – 2004.

Garland, K.J and Noyes, J.M (2004). Computer experience: a poor predictor of computer attitudes, computers in Human behaviour 20(6) pp 823-840

Ghagar, M.N.A, Othman, R., & Mohammedpour 2011. Multilevel Analysis of Achievement in mathematics of Malaysian and Singaporean students. *Journal of educational psychology and counseling vol 2*, pp 285 – 304.

Gisemba, B.J. 2011. The role of teacher characteristics and practice on upper secondary school students' mathematics self efficacy in Nyanza Pronvice of Kenya: a multilevel analysis. *International Journal of science and mathematics education* 9(4), 817 – 842.

Glanz, K; Rimer, B.K & Lenis, F.M 2002.Health Behaviour and Health Education. Theory, Research and Practice. Sam Fracid: Wiley and Sons, Pp 14,17.

Goddard, R. D., Tschannen – Moran, M., & Hoy, W.K. 2001. A Multi – Level Examination of the distribution and effects of teacher trust in students and parents in urban elementary schools. *The Elementary School Journal,* 102, Pp 3 – 17.

Grant, D.M, Malloy, A.D. and Murphy, M.C 2009. A Comparison of Student Perceptions of their Computer Skills to their actual abilities. *Journal of Information Technology Education.* 8, 141-160.

Guzdial, M & Soloway, E. 2002. Log on Education: Teaching the Nintendo Generation to Program.*Communication of the ACM,* 45(4), 17-21.

Hagan, D & Markham, S. 2000. Does it help to have some Programming Experience before beginning a Computer Degree program? *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education,* ACM, NY, 25-28

Harkins, R.J. 2008. The design and implementation of a first course in computer programming for computing majors, non-majors and industry professionals within a liberaleducation frame work. *Information system Educational Journal* 6(60).

Harrison, A.W and Rainer, K. 1992. The Influence of Individual Differences on Skill in End user Computing. *Journal of Management Information Systems,* 9(1), 93-111.

Hasan, B. 2003. The Influence of Specific Computer Experiences on Computer Self Efficacy Beliefs. *Computers in Human Behaviours 19, 443-450.*

Hill, T., Smith, N.D. & Mann, M.F. 1987. Role of efficacy expectations in predicting the decision to use advanced technologies: Case of computers. *Journal of Applied Psychology,* 72(2), 307 – 313.

Hoskey.A.,& Maurino. P.S.A. 2010. Beyond Introductory Programming: Success factors for Advanced Programming. Proceedings of the Information System Educators Conference Nashville Tennesse, USA.

Houle, C. 1996. *The Design of Education 2nd ed.* San Francisco: Jossey-Bass, 173-235.

Howie, S. 2003. Languages and other background factors affecting secondary pupils' performance in Mathematics in South Africa.*African Journal of Research in SMT Education*,7, 1-20

________ 2006. Multi- level Factors Affecting the performance of South African Pupils in Mathematics. In S. Howie & T. Plomp (Eds.), *Contexts of Learning Mathematics and Science* (157 - 176): Routledge.

Hox, J.J. 2002.Multileve analysis techniques and applications. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Hoxmeier, J.A, Nie, W. and Purvis, G.T. 2000.The Impact of Gender and Experience on User Confidence in Electronic Mail.*Journal of End user Computing.* 12(4), 11-20.

Hsiao, H.C., Lin Y.R. and Tu, Y.L. 2010.Gender Differences in Computer Experience and Computer Self-Efficacy among High School Teachers. A paper Presented at the Second Asian Conference on Education, Saka, Japan.

Jegede P.O.2007. Factors in computer Self-Efficacy among South-Western Nigeria College of Education.*Teachers Journal of Psychology in Africa* 17(1).

_________2009a . Predictors of Java Programming self-Efficacy among engineering students in Nigeria University. *International Journal of Computer Science and information security* (IJCSIS), available at http:/site.google.com/site/ijcsis/.

__________ 2009b.Locus of Control and Computer Programming skills acquisitioin among engineering students in a Nigeria university.*Proceedings of 8th Internet Education Conference, Cairo, Egypt*

_________ *2009c.* A study of computer programming preparation of Engineering undergraduates in a Nigerian University. A Paper presented at the 1st National Conference of the Faculty of Technology, Obafemi Awolowo University, Ile-Ife, Nigeria, Volume 1

.

Jenkins, T 2001.The Motivation of Students of Programming. A Thesis Submitted to the University of Kont at Canterbury in the Subject of Computer Science for the degree of Master of Science. 1- 5

Joe, V.C. 1971, "Review of the Internet-External Control Construct as a Personality Variable," *Psychological Reports*, 28, 619-640.

Junge, M.E., & Dretze, B.J. 1995. Mathematical self efficacy gender differences in gifted/talented adolescents. *Gifted child quarterly,* 39, 22-28.

Karsten, R., and Roth, R.M. 1998. Computer Self Efficacy: A Practical Indicator of Student Computer Competency in Introductory Information System Courses. *Information Science,* 1(3), 61-68.

Kay, R. H. 1989. A Practical and Theoretical Approach to Assessing Computer Attitudes: The Computer Attitude Measure (CAM). *Journal of Research on Computing in Education*, 21, 457-463.

Keeves, J. P., & Sellin, N. 1990.Some problems of analysis.*InternationalJournalof Educational Research, 14, 219- 224.*

Kenney – Benson, G.A.,Pomerantz, E.M., Ryhan, A.M., & Patrick .H. 2006. Sex differences in Math performance: The role of children's approach to school work. Developmental Psychology, 42, 11-26.

Knuth D.E & Pardo L.T. 1976.The Early Development of Programming.Computer Science Department, school of Humanities and sciences.Stanford University.

_________ 1976: Ancient Babylonian algorithms, Comme.ACM 15 (1976) 671-677. Encarta in com.ACM19 (1976)108

Koohang & Byrd, D. M. 1987.. A study of Attitudes Towards the Usefulness of the Library Computer System and selected Varaibles: A further study. Library and Infromation Science Research 9 (2): 105 – 111.

Kreft, I, G, G., & Deleeuw, J. (1998). Introducing multilevel modelling.Sage publications

Lefcourt, H.M.1972, Recent Developments in the study of Locus of Control.In B.A. Maher (Ed.), *Progress in Experimental Personality Research*, Volume 4. New York: Academic Press.

___________1992. Durability and Impact of the Locus of Control Construct, *Psychological Bulletin*, 112 (November), 411-414.

Levenson, H. 1974. "Activism and Powerful others: Distinctions within the concept of Internal-External Control". *Journal of personality Assessment*, 38, 377-383.

Lin, M.C. 1985. Gender Equity in Computer Learning Environments.*Computers and the Social Sciences*, 19-27

Mancy, R &Reid, N. 2004. Aspects of Cognitive Style and Programming. A paper presented at the 16th Workshop of the Psychology of Programming Internet Group, Carlow, Island.

Mannila, L., & De Raadt M. (2006). An objective comparison of languages for teaching introductory programming. Proceedings of the 6th Baltic Sea conference on computing research pg 32-37.

Mannila, L., Peltoma Ki, M., and Salakoshi, T (2006). What about a simple language? Analysing the difficulties in learning to program. Computer Science Education 16(3), 2006, 211-228

Marakas, G.M, Yi, M.Y., and Johnson, R. 1998. The Multilevel and Multifaceted Character of Computer Self Efficacy: Toward a Clarification of the Construct and an Integrative Framework for Research. Information Systems Research, 9(2), 126-163.

McGraw – Hill. The history and Evolution of Java.Downloaded from www.books.mcgraw-hill.com.

McKinney, S & Denton L.F (2004). Houston, we have a problem: there is a leak in the CSI affective oxygen tank. Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. *ACM Press, NY,* 236-239.

McNamarah, S. & Pyne, R. 2004. Teaching a first level programming course: strategies for improving students performance, *Journal of Art science and technology, 1.,*pp 42 – 49

Meece, J.L., Ghenke, B.B., & Burg. S. 2006. Gender and Motivation. *Journal of School Psychology,* 44, 351 – 373.BETA programming Language, Addison – Wesly/ ACM Press, 1993.

Metz, S.S. (2007). Attracting the engineering of 2020 today. In R. Burke & M.Matlis (Eds.), Women and minorities in science, technology, engineering and mathematics: upping the numbers pp 184-209. Northampton, M.A: Edward Elgar Publishing.

Mohammadpour, I., Moradi, G.F., & Naijib Abdul Ghafar, M. 2009. Modeling affecting factors on mathematics performance for Singapore eight – grades students based on TIMSS 2007. Paper presented at the proceedings of 2009 International Conference on Social Science and Humanities (ICSSH 2009). Singapore.

Momanyi J.M., Ogoma S.O., Misigo B.L (2010). Gender differences in self-efficacy and academic performance in science subjects among secondary school students in Lugari district.

Myloy, L.P. & J.K. Burton 1988. The Relationship of computer Programming and Mathematics in secondary students. Computers in the schools; 4(3/4): pp 159-166.

Nhundu T.J. 1994. Facet and overall satisfaction with teaching and employment conditions of teachers in Zimbabwe. *Zimbabwe Journal of Educational Research,* 6: 152-194

Norwawi, N.M., Hibadullah, C.F., and Osman, J. 2005. “Factors Affecting Performance in Introductory programming”. [CDROM]. In Proceedings of the International Conference on Qualitative Sciences and Its Applications (ICOQSIA)

Nezlek, J.B. (2001). Multilevel random coefficient analyses of event and interval contigent data in social and personality psychology research. Personality and Social Psychology Bulletin, 27, 771-785.

__________(2007). Multilevel modelling in research on personality. In R.Robins, R.C, Fraley, & R. Krueger (Eds.) Handbook of research methods in personality psychology (pp. 502-523) New York: Guilford.

___________2008. An Introductory to Multilevel Modelling for Social and Personality psychology..*Social and Personality Psycholofy Compass*2 (2) pp 842 – 860.

Nourbachsh, I., Hammer, E., Crowley, K., and Wilkinson, k (2004). Formal measures of learning in a secondary school mobiterobotics contest. *In JEEE international conference on robotics and automation (ICRA)*

Ogunkola, B.J. 2008. Computer Attitude, Ownership and use as predictors of computer Literacy of science Teachers in Nigeria.*International Journal of Environmental & Science Education, Australia* pp 53 – 57. (in press)

Pajares, F. 1997. Current directions in self efficacy research.In H.W Marsh, R.G. Graven & D.M Mclnerney (Eds). International Advances in Self research (pp 1 - 49). Greenwich, Connecticuit: Information Age Publishing.

Park.C.,& Park. D. 2006. Factors Affecting Korean Students' Achievement in TIMSS 1999. In S. Howie & T. plomp (Eds.), Contexts of Learning Mathematics and Science: Routledge. Parsimonions measure. Humanfactors,32, 477 – 491.

Parker, K.R., Chao, J.T., Ohaway, T.A., and Chang, J. (2006). A formal programming language selection process for introducing courses. *Journal of Information Technology Education*, (5), pp. 133-151.

Peyton – Jones, S., Blackwell, A. Burnett, M. A 2003 user – centered approach to functions in Excel.*Proceedings of the 8th ACM* SIGPLAN *international conference on functional programming, (Uppsula, Sweden, August 25 – 29, 2003),* ACM, NY, 165 – 176.

Pintrich , P.R, and De Groot, E.V 1990. Motivational and Self regulated learning components of classroom academic performance. *Journal of Educational Psychology,* 82 (1), 33-40

Pioro B.T 2004.Performance in an Introductory Computer Programming Course as a Predictor of future Success for Engineering and Computer Science majors. A Paper delivered at the International Conference on Engineering Education at Gaines-Ville, Florida October 16-21, 2004.

Potosky D. 2002 A Field study of Computer Efficacy beliefs as an Outcome of Training: the Role of Computer Playfulness, Computer Knowledge, and Performance during Training. *Computers in Human Behaviour,* 18, 241-255

Ramalingam, V. and Wiedenbeck, S. 1998 Development and Validation of Scores on a Computer Programming Self-efficacy Scale and Group Analysis of Novice Programmer Self-efficacy. *Journal of Educational Computing Research,* Vol. 19 (4), .37-386.

______________ LaBelle, D. and Wiedenbeck, S 2004.Self efficacy and Mental Models in Learning to Program. The 9th annual SIGCSE Conference on Innovation and Technology in Computer Science, Leeds, United Kingdom, pp 171 – 175.

Ramirez, M. J. 2006. Factors Related to Mathematics Achievement in Chile. In S. Howie & T. plomp (Eds.), Contexts of learning Mathematics and science: Routledge.

Randell B(1973). The origins of Digital Computers.Selected papers. Berlin: Springer.

Raudenbush, S.W., and Bryk, A.S 2002. Hierarchical Linear models: Applications and data analysis methods. Thousand Oaks, CA: Sage.

Rice, N 2001.Binomial Regression. In Leyland AH, Goldstein H (eds) multilevel modelling of Health Statistics. Wiley, Chichester, Pp 27 – 44.

Robins A, J. Rountree & N. Rountree 2003. Learning and Teaching programming: A review and Discussion. Computer Science Education 13(2), Pp. 137-172.

Rosson, M.B., Ballin, J., and Nash, H. 2004 Everyday programming: challenges and opportunities for informal web development. *Proceedings of the IEE symposia on Visual Languages and Human – Centric Computing,* (Rome, Italy, September 26 – 29, 2004), IEEE Press, , 123 – 130.

Rothermel, G., Burnett, M., Li, L., Dupuis, C., and Sheretov, A. 2001.A methodology for testing spreadsheets.*ACM Transactions on Software Engineering and Methodology*, 10, 1, 110 – 147

Rotter, J. B. 1966, "Generalized Expectancies for Internal Versus External Control of Reinforcement," *Psychological Monographs,* 80, (1, whole no. 609)

____________1990, "Internal Versus External Control of Reinforcement: A case History of a Variable," *American Psychologist,* 45 (April), 489-493

Rural Education Action Project (REAP). (2010). Ownership, Access and Use of computers, information Technology and other e-technologies by students in suburban Beijing schools.Standford University and Chinese Academy of Sciences.

Salamon, G. & Perkins, D. N. 1987. Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research, 3*(2), 149-169.

Sammet J.E 1972. Programming Languages: History and Future.Association for computing machinery, inc.Scale and the Impact of Computer Training.*Educational and Psychological Measurement,* 54(3): 813-821

Scharzer, R 2004."General Perceived Self Efficacy in 14 Cultures" user page fu-berlin.de/~health/selfscal.htm.

Schildt, H (1995). C++ the complete Reference III edition, TMH (TATA McGraw Hill)

Schonberg, E and Drivar, R. (2008). A principled approach to software engineering Education or Java considered harmful. *Ada User journal, 29)(3).*

Schunk, D.H., & Lilly, M.W. (1984). Sex differences in self efficacy and attributions:

Schunk, D.H., & Meece J. L. 2006.Self efficacy development in adolescence.In F. Pajares, & T. Urban (Eds).Self efficacy beliefs of adolescents (pp 71 - 96). Greenwich, connecticuit: Information Age Publishing.

Schwenkglenks, M.M., 2007. Multilevel modelling in the analysis of observational datasets in the health care setting.Inagural dissertation zur Erlangung der Wirde eines Doktors der Philosophie Vorgelegt der Philosophis – National wissenschaftlichen Fakultat der Universitat Basesscience.New York Prentice hall.

Shukur, Z., Alias, M., Hanawi, S.A and Arshad, A (2003). "Faktor-faktor Kegagalan: Pandangan pelajar Yang mengulang Kursus Pengaturcaraan C", Paper presented in Bengkel Sains Pengaturcaran (ATUR03). Kuala Lumpur.

Siegle, D., & Reis, S.M. 1998. Gender differences in teacher and students' ability. Gifted

Skinner, Ellen A. 1996. "A Guide to Constructs of Control," *Journal of Personality and Social Psychology, 71, 549-570.*

Soloway, E. & Ehrlich,1993. Should we teach students to program? Communications of the ACM, 36(10), 21-24.

Stappleton, L.M., 2006. Using multi – level structural equation modelling techniques with complex sample data (Pp 345 - 383). In G.R Hancock & R.O. Mueller (Eds)., Structural Equation modelling : A secnd course. Charlotte N.C: Information Age Publishing

.

Stemler, S.E 2001. Examining School Effectiveness at the Fourth Grade: A Hierarchical Analysis of the Third International Mathematics and Science Study (TIMSS). Dissertation Abstracts International 62(03A), P. 919.

Stronstrup .B 1979, Bjarne Stoustrup:An intermodule communication system for a distributed computer system. Proc..1st Int'l Conf.On distributed computing system. October 1979. pp412-418.

___________1980a, Bjarne Stoustrup: Classes: An abstract Data Type Facility for the C Language. Bell Laboratories Computer Science Technical Report CSTR-84. April 1980. Also Sigplan Notice January, 1982.

___________1980b, Bjarne Stoustrup:A set C Class for Co-routine Style Programming. Bell Laboratories Computer Science Technical Report CSTR-90. November 1980.

___________1982, Bjarne Stoustrup}:Adding Classes to C: An Exercise in Language Evolution. Bell Laboratories Computer Science internal document. April 1982.

____________1984a, Bjarne Stoustrup: The C++ Reference Manual. AT &T Bell Laboratories Computer Science Technical Report No.108 January 1984. (Written in the summer of 1984).Revised version November 1984.

___________1991, Bjarne Stoustrup: The C++ Programming Language (2nd edition). Ndison Wesley. 1991. ISBN 0-201-53992-6s

___________1995. A history of C++: 1979 – 1991.AT&T Bell Laboratories.Murray Hill, New Jersey 07974 www.stroustrup.com/hopl2.pdf. Last accessed November 7, 2012, 5:05pm

Stubbs, M 2001. Words and phrases: Corpus studies on Lexical semantics, Oxford: Blackwell Publishers Limited.

Sullivan A and Bers M.U (2012). Gender difference in Kindergarteners'robotics and programming achievement. *Internationale Journal Technology Des Educ.Springer.*

Taylor, H. & Mounfield, L. 1989.Exploration of the Relationship between Prior Computing Experience and Gender on Success in College Computer Science.*Journal of Educational Computing Research,* 11(4), 291-306

Terwilliger, J.S., & Titus, J.C. 1995. Gender differences in attitudes and attitude changes, The 35th SIGCSE technical symposium on computer science education, Norfolk.

Torkzadeh, G., & Koufterous, X. 1994. Factorial Validity of a Computer Self Efficacy Scale and the Impact of Computer Training.*Educational and Psychological Measurement,* 54(3): 813-821

Van den Broeck, A., Van Danme, J., & Opdenakker, M. C. 2006.The effects of student, Class and School Characteristics on TIMSS 1999 Mathematics Achievement in Flanders. In S. Howie & T. plomp (Eds.), Contexts of learning Mathematics and Science: Routledge.

Van der Westhuizen P.C & Du Toit S.C. 1994. Werksbevrediging by die swart onderwysers. *South African Journal of Education*, 14: 145-149

Van Dyke, C. (May 1987). Taking "Computer Literacy" literally. Communications of the ACM, 30(5), 366 – 374.

Vekiri, I., and Chronaki, A. (2008). Gender issues in Technology use: Perceived Social Support, computer self efficacy and value beliefs, and computer use beyond school. Computers and Education, 51(3), 1392 – 1404.

Verkeyn, A 2005. History of Programming Languages.Last Accessed: November 7, 2012, 5:05pmVirginia, USA: ACM Press.

Wahab, M.H.A, Farhan, m.m.m., Norwawi, N.M., Hibadullahi, C.F., Zaiyadi M.F. 2010.An investigation into influence factor of student programming Grade using Association Rule Mining. Advanced in information sciences and service sciences. Vol 2 No 2, Pp 19 – 27

Walkey, F.H. 1979. “Internal Control, Powerful others, and chance: A Confirmation of Levenson’s Factor Structure,” Journal of Personality Assessment, 43(5), 532-535.

Wang, C. And Bird, J.J 2011. Selecting statistical procedures for multi – level Data: examinig relationship between principal authenticity and teacher trinst and Engagement. A brief report services of the centre for Educational measurement and Evaluation. Department of Educational Leadership, UNC Charlotte.

Wexelblat, Richard L., 1981(ed.) History of programmiing Languages. New York:Academic Press

Wiedenbeck, S. & Ramalingham, V.and Engerbretson, A. 2004. Comprehension strategies of end – user programmers in event driven application.*Proceedings of the IEEE Symposia on Visual Languages and Human – Centric Computing,* (Rome, Italy, September 26 – 29, 2004), IEEE Press, 2004, 207 – 214.

________________LaBelle.D & Kain, V.N.R. 2004.Factors Affecting Course Outcomes in Introductory Programming.Proceedings of the 16th Workshop of the Psychology of Programming Interest Group. Carlow, Ireland, 97 -110.

_______________ 2005. Factors affecting the success of non-majors in learning to program.Proceedings of the first International Workshop of Computing Education Research. Seatle, 13-24.

Wilson, B. C 2000.Contributing factors to success in computer science: A study of Gender differences. A dissertation submitted in partial fulfillment of the requirements for the Doctors of Philosophy Degree of the Department of Curriculum and instruction in the Graduate school, Southern ilinois

_____________& Shrock S. 2001.Contributing to success in an Introductory Computer Science Course: a study of twelve factors.Proceedings of the 32nd SIGCSE technical symposium on computer science Education. ACM press, NY, pp 184 – 188

_____________2002.A study of Factors Promoting Success in Computer Science including Gender Differences.*Computer Science Education* Volume, 12, No 1 – 12, pp 141 – 164.

Witt – Rose D.L (2003). Student Self Efficacy in College Science: An investigation of geder, age and Academic Achievement. A Research Paper Submitted to the Graduate School, Universty of Wisconsin-Stout in Partial Fulfilment of the Requirements for the master of Science degree with a major in Education

Woodhouse, G. Goldstein, H 1999. Multilevel statistical models.1st Internet Edition.Edward Arnold London.

Yukselturk, E. & Bulnut, S. 2007. Predictors for student success in an online course. Educational Technology & Society, 10(2), Pp 71 – 83.

Zimmerman, B.J, & Schunk, D.H 2003. Albert Bandura: The Scholar and his Contributions to educational psychology: A century of contributions (pp 431 – 457). Mahwah, NJ. L. Erlbaum Associates.

Zuse K 1976. Kommentar Zum Plankalkul in Berichte der Gesellschaft für Mathematik und Daten verarbeitung. No. 63 (Bonn, 19767), 21 – 41

APPENDIX I

COMPUTER BACKGROUND QUESTIONNAIRE

Thank you for your willingness to participate in this study. Your individual responses to this questionnaire will be kept confidential, and will be destroyed after the collective data is gathered and analysed. Please be as honest and accurate as you possibly can, so that the data given to the researcher can be relied upon for research purposes.

Demographic & Background Data:

Gender: Male ______ Female _______ (Tick on.

Ownership of Computer: Yes ______ No _____ (Tick one)

Mathematics Background:

(a) Did you take Further Mathematics in WASSCE/NECO/NABTEB etc?

Yes___ No ____ (Tick one)

(b.) Indicate how many Mathematics courses you took in the University before entering the Java class ______

Background in C++

Have you taken a course in C++ before? Yes ________ No _______

Please indicate other programming languages you learnt before entering Java class

APPENDIX II

COMPUTER EXPERIENCES SCALE

Indicate your level of experience (prior to taking the Java programming language course) with the following on a ten-point interval from 1 (no experience) to 10 (very experienced).

i. Word processing experience (eg Microsoft Word) ______

ii. Spreadsheet experience (eg Microsoft excel) _____

iii. Database experience (eg Microsoft Access) _____

iv. Presentation Software experience (e. g Microsoft Power Point) _____

v. Operating System experience _____

vi. Computer Graphics experience ______

vii. Computer Games experience _____

viii. Internet experience ______

ix. Statistical Package experience (eg SPSS, AMOS, MINITAB) _____

x. Programming experience _____

APPENDIX III

JAVAPROGRAMMING SELF-EFFICACY SCALE

Rate your confidence in doing the following Java programming related tasks by filling the number that best represents your level of experience in front of each task using a scale of 1 (not at all confident) to 7 (absolutely confident). If a specific task is totally unfamiliar to you, please fill 1.

Not at all confident	Mostly not confident	Slightly confident	50/50	Fairly confident	Mostly confident	Absolutely confident
1	2	3	4	5	6	7

1. I could write syntactically correct **Java** statements. _______
2. I could understand the language structure of **Java** and the usage of the reserved words. __
3. I could write logically correct blocks of code using **Java** _______
4. I could write a **Java** program that displays a greeting message. _______
5. I could write a **Java** program that computes the average of three numbers. _______
6. I could write a **Java** program that computes the average of any given number of numbers.
7. I could use built-in functions that are available in the various Java **applets**. _______
8. I could build my own **Java applets**. _______
9. I could write a small Java program given a small problem that is familiar to me.
10. I could write a reasonably sized Java program that can solve a problem that is only vaguely familiar to me. _
11. I could write a long and complex Java program to solve any given problem as long as the specifications are clearly defined. _______
12. I can organize and design my program in a modular manner. ____
13. I understand the object-oriented paradigm. ____
14. I can identify the objects in the problem domain and declare, define, and use them. ____
15. I can make use of a pre-written function, given a clearly labeled declaration of the function. ____
16. I can make use of a class that is already defined, given a clearly labeled declaration of the class. ____

17. I can debug (correct all the errors) a long and complex program that I had written and make it work. _____

18. I can comprehend a long, complex multi-file program. ____

19. I could complete a programming project if someone showed me how to solve the problem first. ____

20. I could complete a programming project if I had only the language reference manual for help. ____

21. I could complete a programming project if I could call someone for help if I got stuck. ____

22. I could complete a programming project once someone else helped me get started. ____

23. I could complete a programming project if I had a lot of time to complete the program. ____

24. I could complete a programming project if I had just the built-in help facility for assistance. ____

25 I could find ways of overcoming the problem if I got stuck at a point while working on a programming project. ____

26 I could come up with a suitable strategy for a given programming project in a short time. _

27 I could manage my time efficiently if I had a pressing deadline on a programming project

28 I could mentally trace through the execution of a long, complex, multi-file program given to me. ___

29 I could rewrite lengthy confusing portions of code to be more readable and clear. ____

30 I can find a way to concentrate on my program, even when there were many distractions around me. ____

31 I can find ways of motivating myself to program, even if the problem area was of no interest to me. ____

32 I could write a program that someone else could comprehend and add features to at a later date. ____

APPENDIX IV

LEVENSON LOCUS OF CONTROL

Instruction: Following is a series of statements. Each represents a commonly held opinion. There are no rights or wrong answers. You will probably agree with some items and disagree with others. We are interested in the extent to which you agree or disagree with such matters of opinion. Read each statement carefully, then indicate the extent to which you agree or disagree using the following responses:

If you agree strongly, respond	+3
If you agree somewhat, respond	+2
If you agree slightly, respond	+1
If you disagree slightly, respond	-1
If you disagree somewhat, respond	-2
If you disagree strongly, respond	-3

1. Whether or not I get to be a leader depends mostly on my ability. ______

2. To a great extent my life is controlled by accidental happenings. ______

3. I feel like what happens in my life is mostly determined by powerful people ______

4. Whether or not I get into a car accident depends mostly on how good a driver I am. ______

5. When I make plans, I am almost certain to make them work. ______

6. Of ten there is no chance of protecting my personal interests form bad luck happenings. ______

7. When I get what I want, it is usually because I'm lucky. ______

8. Although I might have good ability, I will not be given leadership responsibility without appealing to those positions of power. ______

9. How many friends I have depend on how nice a person I am. ______

10. I have often found that what is going to happen will happen. ______

11. My life is chiefly controlled by powerful others. ______

12. Whether or not I get into a car accident is mostly a matter of luck. ______

13. People like myself have very little chance of protecting our person interests when they conflict with those of strong pressure groups. ______

14. It is not always wise for me to plan too far ahead because many things turn out to be a matter of good or bad fortune. ______

15. Getting what I want requires pleasing those people above me. ______

16. Whether or not I get to be a leader depends on whether I'm lucky enough to be in the right place at the right. ______

17. If important people were to decide they didn't like me, I probably wouldn't make many friends. ______

18. I can pretty much determine what will happen in my life. ______

19. I am usually able to protect my personal interests. ______

20. Whether or not I get into a car accident depends mostly on the other driver. ______

21. When I get what I want, it's usually because I worked hard for it. ______

22. In order to have my plans work, I make sure that they fit in with the desires of people who have power over me. ______

23. My life is determined by my own action ______

24. It is chiefly a matter of fate whether or not I have a few friends or many friends. _____

APPENDIX V

JAVA PROGRAMMING ACHIEVEMENT TEST

SECTION A: MUTIPLE CHOICE QUESTIONS

Instruction: The Java multiple choice questions is aimed at assessing students' achievement in few of the Java programming language topics. The purpose is strictly for research work only. Your cooperation will be highly appreciated. Under this section; **you are to circle the correct option in each question.**

1. If s = "text", the value returned by s.length ( ) is (a) false (b) true (c) 4 (d) 5

2. What is the value of k after the following code fragment? Int k = 0j; int n = 12 while (k < n) { k = k + 1} (a) 0 (b) 11 (c) 12 (d) unknown

3. Given the following code fragment int A [ ]; int i = 0; A = new int A [4]; while (i < 4) {A [i] = 10 i = i + i;} what is the value of A [ ]? (a) 0 (b) 3 (c) 10 (d) unknown

4. What is the purpose of this bit of code

```
void int ( )
{
  …
}
?
```

(a) a class that initializes the applet (b) a required method in an applet

(c) a place to declare variables (d) interacting with the user

5. A compound statement is: (a) a collection of one or more statements enclosed in braces (b) a statement involving if and else (c) a way of declaring variables (d) a way of setting the value of a variable

6. The method set label can be used with what type of object?

(a) Double field (b) int (c) text field (d) string

7. Consider the following code:

```
int x, y, z;
y = 1,
```

z = 5

x = 0 – (++y) + z++;

After execution of this, what will be the values of x, y and z?

(a) x = 4, y = 1, z = 5 (b) x = -7, y = 1, z = 5 (c) x = 4, y = 2, z = 6 (d) x = 3, y = 2,z =6

8. You want subclasses in any package to have access to members of a superclass. Which is the most restrictive access that accomplishes this objective?

(a) Public (b) private (c) protected (d) transient

9. Which three form part of correct array declarations?

(1) Public int a [ ] (2) static int [ ] a (3) public [ ] int a

(4) Privateint a [3] (5) private int [3] a [ ] (6) public final int [ ] a

(a) 1, 3, 4 (b) 2, 4, 5 (c) 1, 2, 6 (d) 2, 5, 6

10. Given a method in a protected class, what access modifier do you use to restrict access to that method to only the other members of the same class?

(a) final (b) static (c) private (d) protected

11. Which four options describe the correct default values for array elements of the types indicated?

(1) int - > 0

(2) String - > "null"

(3) Dog - > null

(4) char - > '\u0000'

(5) float - > 0.0f

(6) boolean - > true

(a) 1, 2, 3, 4 (b) 1, 3, 4, 5 (c) 2, 4, 5, 6 (d) 3, 4, 5, 6

12. Which one of these lists contains only Java programming language keywords?

(a) class, if, void, long, Int, continue

(b) goto, instanceof, native, finally, default, throws

(c) try, virtual, throw, final, volatile, transient

(d) strictfp, constant, super, implements, do

13. Which is a reserved word in the Java programming language?

(a) method (b) native (c) subclasses (d) reference

14. Which three are legal array declarations?

(1) int [] myScores [];

(2) char [] myChars;

(3) int [6] myScores;

(4) Dog myDogs [];

(5) Dog myDogs [7];

(a) 1,2,4 (b) 2,4,5 (c) 2,3,4 (d) all are correct

15. Which three are valid declarations of a char?

(1) char c1 = 064770;

(2) char c2 = 'face';

(3) char c3 = 0xbeef;

(4) char c4 = \u0022;

(5) char c5 = '\iface';

(6) char c6 = '\uface';

(a) 1, 2, 4 (b) 1, 3, 6 (c) 3, 5 (d) 5 only

16. Which is the valid declarations within an interface definition?

(a) public double methoda();

(b) public final double methoda();

(c) static void methoda(double d1);

(d) protected void methoda(double d1);

17. Which one is a valid declaration of a boolean?

(a) boolean b1 = 0;

(b) boolean b2 = 'false';

(c) boolean b3 = false;

(d) boolean b4 = Boolean.false();

18. Which is a valid declarations of a String?

(a) String s1 = null;

(b) String s2 = 'null';

(c) String s3 = (String) 'abc';

(d) String s4 = (String) '\ufeed';

19. Which is valid declaration of float?

(a.) float f = 1F; (b.) float f = 1.0; (c.) float f = “1”; (d.) float f = 1.0d;

20. Which of the following are Java reserved words?

1. run 2. import 3. default 4 implement

(a.) 1 and 2 (b.) 2 and 3 (c.) 3 and 4 (d.) 2 and 4

SECTION B : ESSAY TYPE QUESTIONS

The following two programs have some errors; identify these errors and rewrite the preceding codes on the dotted lines provided .

PROGRAM 1: A Program to Compute the Factorial of Numbers

```
package testquestions;
import java.util.scanner;

……………………………………….
public Class Factorial {

…………………………………………………………………….
int x;

public Factorial(String number)
   {
      x = number;

     ……………………………………………………………..
   }

private void computeFactorial()
   {
int counter,factorialNumber = 1;

for(counter = 1;counter<x;)

      ……………………………………………………………..
     {
factorialNumber * = counter;

        ………………………………………………………….
```

```
    }//end of forloop

System.out.println("The factorial of %d is %d",x,factorialNumber);

    ……………………………………………………………………………………………………
  }//ends method compute factorial

public static void main(String[] args) {

System.Out.println("Enter the number to find its factorial: ");

     …………………………………………………………………
scanner input = new Scanner(System.in);

     …………………………………………………………….
     String factorialNumber = input.Next();

    ……………………………………………………………
     Factorial objectFactorial = new Factorial(factorialNumber);

objectFactorial.computeFactorial();
  }//ends main method
}// end of class factorial
```

PROGRAM 2: A Program to Compute the Sum of Two Numbers

```
package testquestions;

Import java.Util.Scanner;

……………………………………..
public class SumNumbers {
  Public static void main(string[] args) {

    …………………………………………………….
      Int a,b;
     …………………………………………………..
      Scanner input = Scanner(System.in);

     ……………………………………………………….
System.Out.println("Enter the value of the first  Number: ");

      ………………………………………………..

      a = input.nextInt();

System.Out.println("Enter the value of the Second  Number: ");

      …………………………………………………………………………………………………
      b = input.nextInt();

int c = Sum(a,b);

      ……………………………..
System.out.println("The sum of %d and %d is %d",a,b,c);

…………………………………………………………………………………………..
   }

private static int sum(int a , int b)
   {
return (a+b);
   }
}
```

APPENDIX VI

SOLUTION TO THE JAVA ACHIEVEMENT TEST

SECTION A

1.	**C**
2.	**D**
3.	**D**
4.	**A**
5.	**A**
6.	**D**
7.	**D**
8.	**C**
9.	**C**
10.	**C**
11.	**A**
12.	**B**
13.	**B**
14.	**A**
15.	**A**
16.	**A**
17.	**C**
18.	**A**
19.	**D**
20.	**D**

SECTION B

PROGRAM 1: A Program to Compute the Factorial of Numbers

package testquestions;

import java.util.scanner;

import.java.util.Scanner

……………………………………….

public Class Factorial {

public.class factorial

……………………………………………………………………

int x;

public Factorial(String number)

{

x = number;

x = integer.parseInt (number);

…………………………………………………………………..

}

private void computeFactorial()

{

int counter,factorialNumber = 1;

for(counter = 1;counter<x;)

for (counter = 1: counter < = x; counter ++)

…………………………………………………………………..

{

factorialNumber * = counter;

factorialNumber * = counter

………………………………………………………………

}//end of forloop

System.out.println("The factorial of %d is %d",x,factorialNumber);

System.out.printf (“The factorial of % d is % d” ; x, factorialNumber);

…………………………………………………………………………………………………

}//ends method compute factorial

public static void main(String[] args) {

System.Out.println("Enter the number to find its factorial: ");

System.out.println (“Enter the number to find its factorial: ");

………………………………………………………………

scanner input = new Scanner(System.in);

Scanner input = new Scanner(System.in);

…………………………………………………………….

String factorialNumber = input.Next();

String factorialNumber = input.next();

…………………………………………………………

Factorial objectFactorial = new Factorial(factorialNumber);

objectFactorial.computeFactorial();

}//ends main method

}// end of class factorial

PROGRAM 2: A Program to Compute the Sum of Two Numbers

package testquestions;

Import java.Util.Scanner;

Import java.util.Scanner;

………………………………………..

public class SumNumbers {

Public static void main(string[] args) {

public static void main(string[] args) {

…………………………………………………….

Int a,b;

int a,b;……………………………………………………..

Scanner input = Scanner(System.in);

Scanner input = new Scanner(System.in);

…………………………………………………………

System.Out.println("Enter the value of the first Number: ");

System.out.println("Enter the value of the first Number: ");

………………………………………………..

a = input.nextInt();

System.Out.println("Enter the value of the Second Number: ");

System.Out.println("Enter the value of the Second Number: ");

…………………………………………………………………………………………………

b = input.nextInt();

int c = Sum(a,b);

int c = sum(a,b);

……………………………..

System.out.println("The sum of %d and %d is %d",a,b,c);

System.out.printf ("The sum of %d and %d is %d",a,b,c);

……………………………………………………………………………………………..

}

private static int sum(int a , int b)

{

return (a+b);

}

}