UNIVERSITY OF SHERBROOKE

LES FACTEURS CONTRIBUANT A LA RÉUSSITE DANS LES COURS D'INTRODUCTION EN PROGRAMMATION

FACTORS INFLUENCING SUCCESS IN INTRODUCTORY PROGRAMMING COURSES

par

Pit Foung Lan Chow Wing

Essai présenté à la Faculté d'éducation

En vue de l'obtention du grade de

Maître en éducation (M.Éd.)

Maîtrise en enseignement au collégial

Novembre 2009

UNIVERSITÉ DE SHERBROOKE

Faculté d'éducation

Maîtrise en enseignement au collégial

Les facteurs contribuant à la réussite dans les cours d'introduction en programmation

Factors influencing success in introductory programming courses

Par

Pit Foung Lan Chow Wing

a été évalué par un jury composé des personnes suivantes:

______________________________ Silke Lach	Directrice de l'essai
______________________________ Elizabeth Charles	Évaluatrice de l'essai

SUMMARY

Learning programming is not an easy task. Research so far has shown that computer science students in general lack basic programming skills. Half of those in university introductory programming courses do not make it to the next level. Several studies have looked at different factors that could be keys to success in programming. Factors that were most commonly examined were prior computer courses, prior academic background, level of mathematics knowledge, programming skills and number of hours of study. Results of those studies did not give a clear answer as to what leads to success in programming courses.

This study looked at the reasons why Vanier College students in computer programming are encountering difficulties in their learning process. Factors such as prior academic background, prior computer experience, mother tongue, and learning styles were examined to see how they play a role in students' success in programming courses. The research of Booth (1992) and Bruce et al. (2001) informed this study. Booth did a phenomenographic qualitative study on learning to program by interviewing computer science students in their first year of studies. Booth found out that the students' learning methods can be grouped into four categories: coding, understanding & integrating, problem solving and participating & enculturation. Bruce et al. added one more category: following. However, these researchers did not look at the relationship between ways of learning and student success. This study was an attempt to see whether Vanier College students learning programming can be categorized according to Booth and Bruce et al. Furthermore, it tried to see whether success depend on learning styles. The initial research hypotheses were the following:

- *Computer Science students using understanding and integrating succeed better than students using following, coding, or problem solving.*

- *Students using problem solving succeed better than those who use participating and enculturation.*
- *Students who use coding perform better than those who prefer participating and enculturation.*

In addition, this study hoped to examine whether there is a gender difference in how students learn programming.

Both quantitative and qualitative methods were used in the study. Quantitative data was collected via a survey of fifty-eight Vanier College students. Qualitative data was generated by an open-ended question in the survey as well as by personal interviews with ten computer science students. The statistical package SPSS was used to analyze the quantitative data. The qualitative data was analyzed using content analysis.

Only eight female students took part in the survey. With such a small proportion of females, gender could not be considered as a factor in this study. The data also showed that most of the students (43 out of the 58) said that they used either coding, understanding and integrating or problem solving as their learning style. Only 11 considered that they used participating and enculturation or following as their way of learning. Correlation analyses were done using Spearman's rho and Kendall's tau correlations. They showed that high school average and high school math grade did have a slight positive effect of the final mark that the students received but the relationship was not significant. However, there was a significant positive relationship between high school average and high school math grade and the midterm mark. Furthermore, prior computer knowledge and prior basic programming knowledge play a positive role in success in learning how to program. The less prior computer knowledge and the less prior basic programming experience the students have, the lower their final marks were.

Since the number of students in the two latter learning style categories is statistically insignificant and furthermore, since Spearman's rho and Kendall's tau correlation showed that there was no significant correlation between ways of learning and final marks of the students, these results led to a revision of the initial objectives of the project. It cannot be said that students using coding succeed better than those using understanding & integrating or problem solving.

The qualitative data analysis aimed to examine Vanier students' ways of learning how to program, and to see whether these fit the categories of Booth (1992) and Bruce et al. (2003). The qualitative data strongly support the findings of the quantitative data. Three kinds of observations could be made from the interviews. The interviews revealed details about the learning process of the students, the difficulties the students encountered and the ways they coped with the difficulties. The interviews confirmed that Vanier College computer science students learn programming in mainly three ways: coding, understanding and integrating, and problem solving.

A phenomenon that emerged in the interviews was that some students used rote learning in their programming courses. They tried to learn the concepts and the syntax of the programming language by heart without trying to understand how to apply them. They just learned the situations where they could apply the concepts and rules. This can be counterproductive to their learning process in that they may get disoriented when they encounter unfamiliar problems or situations. Another observation made from the interviews can be considered of importance. Three of the ten students complained about attitudes of their teachers. They said that they did not get much help from the teachers when they needed it. This led to increasing disinterest in the subject and their dropping out of the course or program. It would therefore be pertinent to look at the various ways computer programming is being taught and how the learning styles can better accommodate Vanier College's computer programming students.

SOMMAIRE

Apprendre la programmation n'est pas une tâche facile. Jusqu'à date, les études ont démontré que les étudiants en informatique n'ont pas, en général, les compétences de base nécessaires pour programmer. De ceux qui ont commencé les études préliminaires en programmation, la moitié n'a pas atteint le niveau adéquat pour continuer en deuxième année. Plusieurs études ont examiné différents facteurs qui pourraient contribuer à la réussite en programmation. Les facteurs les plus souvent examinés étaient les précédents cours en informatique, les études académiques précédentes, le niveau de connaissance en mathématiques, les compétences en programmation et le nombre d'heures d'études. Les résultats de ces études n'ont pas donné une réponse claire à ce qui amène à la réussite dans les cours de programmation.

La présente étude a examiné les raisons pour lesquelles les étudiants en informatique du Collège Vanier rencontrent des difficultés dans leurs études en programmation. Les facteurs tel que le niveau des études précédentes, l'expérience en informatique, la langue maternelle et les méthodes d'apprentissage ont été considérés pour voir quel rôle ces facteurs jouent pour promouvoir la réussite dans les cours de programmation. Cette étude est basée sur les travaux de Booth (1992) et de Bruce et al. (2001). Booth a fait une étude qualitative sur l'apprentissage en programmation en interrogeant des étudiants en première année d'études en informatique. Booth a constaté que les méthodes ou styles utilisés par les étudiants peuvent être regroupés en quatre catégories : le codage, la compréhension et l'intégration, la résolution des problèmes, et la participation dans la culture informatique. Bruce et al. (2001) a ajouté une autre catégorie : « suivre ». Cependant, ces chercheurs n'ont pas considéré les relations entre les styles d'apprentissage et la réussite dans les cours de

programmation. La présente étude a essayé de voir si l'apprentissage en programmation des étudiants du Collège Vanier pourrait être catégorisé selon Booth et Bruce et al. De plus, l'étude a essayé de voir si leur réussite dépendait des styles d'apprentissage. Les hypothèses initiales de recherche ont été formulées comme suit :

1. *Les étudiants en informatique utilisant la compréhension et l'intégration réussissent mieux que ceux utilisant « suivre », le codage ou la résolution des problèmes.*
2. *Les étudiants utilisant la résolution des problèmes réussissent mieux que ceux qui utilisent la participation dans la culture informatique.*
3. *Les étudiants utilisant le codage réussissent mieux que ceux qui utilisent la participation dans la culture informatique.*

De plus, la présente étude espérait examiner s'il y a une différence de performance entre les deux genres.

Les méthodes quantitatives et qualitatives d'analyse sont utilisées pour l'étude. Les données quantitatives étaient recueillies par un sondage vis-à-vis cinquante-huit étudiants du Collège Vanier qui prenaient le cour d'introduction à la programmation. Ces données ont été analysées par le programme de statistiques SPSS. Les données qualitatives ont été récupérées d'une question ouverte se trouvant dans le questionnaire et aussi des entrevues personnelles avec dix étudiants en informatique. Ces données qualitatives ont été analysées par la méthode de l'analyse de contenu.

Huit filles seulement ont pris part au sondage. Avec cette minime quantité de participantes, le genre ne pourrait pas être pris en considération dans l'étude. Aussi, la plupart des cinquante-huit participants (43 au juste) ont affirmé qu'ils utilisaient le codage ou la compréhension et l'intégration ou la résolution des problèmes en tant que style d'apprentissage. Les onze autres ont affirmé qu'ils utilisaient la participation dans la culture informatique ou « suivre ». La corrélation rho de Spearman et celle de tau de Kendall ont été utilisées pour fin d'analyse de corrélation. Ces analyses ont démontré que la moyenne au secondaire et la note du cours des mathématiques au secondaire avaient un effet insignifiant sur la note finale en programmation. Toutefois, il y avait une corrélation significative enter la moyenne au secondaire et la note des mathématiques, et la note à l'examen de mi-session. De plus, la connaissance en informatique et en programmation précédant le premier cours de programmation avaient un effet positif sur la réussite en programmation. Plus l'étudiant possédait de la connaissance en informatique avant le cours, plus la note finale était haute.

Vu que le nombre des étudiants qui considéraient qu'ils utilisaient les deux derniers styles d'apprentissage était statistiquement insignifiant et, de plus, le rho de Spearman et le tau de Kendall ont démontré que la corrélation entre les méthodes ou styles d'apprentissage avec la réussie n'était pas significative, ces résultats ont poussé à la révision des objectifs initiaux du projet. On ne pourrait pas dire que les étudiants utilisant le codage réussissent mieux que les étudiants utilisant la compréhension et intégration ou la résolution des problèmes.

Les données qualitatives étaient là pour examiner de plus près les méthodes d'apprentissage en programmation des étudiants du Collège Vanier, et pour voir si ces méthodes correspondent à celles de Booth (1992) et de Bruce et al. (2001). Ces données ont confirmé solidement les résultats obtenus des données quantitatives. Trois types d'observations pouvaient être faits des entrevues. Ces entrevues ont relevé la façon dont les étudiants apprenaient la programmation, les difficultés qu'ils ont rencontré, et comment ils ont pu résoudre ces difficultés. Aussi, les entrevues ont confirmé que les étudiants du Collège Vanier utilisaient surtout trois méthodes d'apprentissage : le codage, la compréhension et intégration, et la résolution des problèmes.

Un phénomène a émergé des données émanant des entrevues. Quelques étudiants apprenaient la programmation en mémorisant les concepts et la syntaxe du langage de programmation sans comprendre comment les utiliser. Ils retenaient par cœur seulement les situations où ils pourraient utiliser les concepts et les règles. Cette situation peut aller à l'encontre du but recherché. Ils se retrouveraient désorientés et confus devant des situations et des problèmes qu'ils n'ont jamais rencontrés. Une autre observation très importante est le fait que trois des participants des entrevues se sont plaints des attitudes négatives de certains professeurs. Ils disaient qu'ils n'avaient pas assez d'aide de ces professeurs quand ils en avaient besoin. Ils sont devenus désintéressés et ont pensé à abandonner le cours ou le programme. Il serait pertinent de voir les différentes stratégies d'enseignement de la programmation et comment ces méthodes pourraient être utilisées pour accommoder les exigences des étudiants en informatique du Collège Vanier.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

It is a well-known fact that programming courses are considered difficult by students. Teachers have been and are still complaining about the poor performance of students in the introductory programming courses. Surveys that have been done so far show that failure rates in introductory programming courses are very high. Students in our English CEGEP (Collège d'Enseignement Général Et Professionnel) computer science programs including Vanier College are no exception; they are also struggling to succeed in these courses. They have difficulties writing an algorithm, the basis of all programming problems. At the beginning of the first semester, the majority of new students have the misconception that they will do well in programming because they have been playing with computers for a long time. After some time in the first programming course, they realize that programming is not as easy as they have thought it would be. They forget that computers are machines and will perform and process only what is being asked of them. They have difficulties in learning "logical thinking", an important requirement for succeeding in a programming course. Another related problem is that they often cannot apply and combine the various concepts in programming that they are taught in order to solve programming problems. Most of them are surprised to learn that the tasks to be programmed have to be very detailed so that an optimal solution could be found. There are also other important misconceptions (which will be later discussed in the literature review) that hinder novice students.

1. PROBLEM STATEMENT

Although programming is one of the many skills that computer science students must acquire, there have been concerns expressed by computer science teachers about their students lacking basic programming skills. Poor performance and high failure rates in introductory programming courses are common. McCracken (2001) headed a working group made up of researchers from five countries that studied the phenomenon. They devised assessment exercises to test the programming skills of first-year computer science students from four universities. The team reported that many students do not know how to program at the end of their introductory programming courses. Two hundred and sixteen students took the assessment test and the average score was just 20.8%.

In another study it was found that half of the students in the Department of Computer Science at the University of Glasgow do not achieve the minimum grade necessary to automatically progress to the next level (Mancy & Reid, 2004). In 2002-2003, only 50% of students obtained a grade C or better which is the required grade to continue to the next level. The figure confirms that students commonly experience difficulties with programming.

At Vanier College, during the academic years 2002 to 2006, an average of 32% of the students in the Computer Science Program failed their first programming course, although emphasis was being placed on supervised practical exercises in laboratories, which is usually not done at universities. The figure was calculated from success data extracted from the SRAM (Service Régional d'Admission du Montréal métropolitain) database.

Thus, we need to know more about how the learning of programming by computer science students takes place, how students come to understand concepts and gain the ability to apply these concepts, and how they gain the technical and practical skills needed to be able to write efficient programs. We also need to understand why failure rates among programming students are so high by considering numerous factors such as prior academic background of the students, time spent studying and practicing programming, and regular class attendance. Do computer science students succeed better if they have a strong mathematics background? We need to look at students' abilities and methods of processing information that they receive in introductory programming courses about programming concepts, programming logics and techniques, and the syntax of the programming language used.

Since most studies done to date were done at the university level, it is interesting to see whether similar results are obtained at the CEGEP level. This study aimed to find out how our CEGEP students are really coping with their programming courses and to learn about their concerns. This research should therefore benefit students as well as teachers. The students struggling with introductory courses may require particular pedagogical approaches on the part of their teachers. It is hoped that teachers can then better understand the state of mind of the students in their learning process and become more aware of the difficulties that students are encountering. At the same time certain misconceptions can be clarified. Teachers will then be able to modify their instructional strategies to make the courses more interesting and motivating to the students and help them in their learning process. In a large sense, improving the quality of teaching and learning was the main aim of this research.

CHAPTER 2

LITERATURE REVIEW

Scholarly articles investigating how students are doing in computer science have been scarce until recently. Computer science teachers are rarely interested in educational research since most of them do not have formal teaching training. Most of the articles published by teachers on this subject focus on their course contents and teaching practices. But learning programming is not a trivial task. This chapter introduces a model of learning most appropriate to programming followed by some results of previous research done on learning and teaching of the subject.

1. CURRENT LITERATURE

1.1 A Current Model of Learning

Every student who starts a computer science program or any other related field has some kind of knowledge and background. This knowledge has been actively constructed by the student through different means. Programming cannot be learned by passively absorbing materials from textbooks and lectures. Rote learning also does not help. Learning programming must follow a constructivist approach. This approach is a dynamic one; students construct more knowledge using their prior knowledge. This can be done through much practice, writing, testing and running programs. The students make progress by learning from the errors they make in their programs. Students may learn in different ways, but they all should know the programming concepts and how to use them in their tasks in order to succeed. There are different factors that contribute to their ability to apply these concepts to successfully write a program that is efficient and at the same time produces what it is supposed to produce. This study looks at some of the factors that contribute to success in introductory programming courses at Vanier College.

1.2 Misconceptions

Early research considering the issues involved in learning how to program looked at the cognitive aspects of the problem. Du Boulay (1989) analyzed the areas in which students had difficulties. The areas include understanding the properties of the machine, knowledge of the syntax of programming languages, understanding the structures of programs as well as knowledge of the editors used to enter programs in the computer, correct them for compiling errors, test and execute them. Du Boulay also analyzed several misconceptions that novices have about programming concepts. Most of the examples used in explaining concepts are mathematically oriented, but students cannot see the difference between a variable in algebra and the notion of a variable in programming. They cannot see how a variable or a variable name represents an address of the computer's memory. They were surprised to see that in programming it is possible to write the following statement (in BASIC, a programming language in the early days of computer science): A = A + 1 meaning that you are assigning the result of the increment by one of the variable A to A itself. In Algebra, this statement is illegal. Also, students cannot see the logic behind the statement (in PASCAL, another popular programming language mainly used to teach elements of programming) A := B; Novices would see a link between A and B such that whatever happens to A will also happen to B, which is wrong; the other way round is right. These are two misconceptions among others that may result in students making errors in their programming tasks. Furthermore, anecdotal evidence shows that some students think that they would be good in programming because they have been using computers playing games and do word-processing since an early age. But when it comes to programming they have difficulties reading and interpreting error messages when they try to compile and run their programs, and students are weak in taking appropriate action.

1.3 Cognitive abilities in programming

Linn & Dalbey (1989) surveyed 500 high school students looking at what they called a chain of cognitive accomplishments needed in programming and studying how the chain leads to progress in the learning of the subject. That chain consists of the language features of the programming language used, design skills and problem-solving abilities of the students. The language features are the rules or syntax of the language. These are usually explained to the students by the teachers who give exercises so that the students are able to see where and how to use them.

Design skills are the next step that the students need to grasp. They are the set of techniques that combine the language features to write a program to solve a problem. These techniques are used by the students to create a series of templates, each of which performs a particular function. Design skills also involve planning, testing and reformulating. Planning is needed to solve complex programming problems and this is rarely done by novice programmers. On the other hand, testing is important in order to know whether the programs perform as they are intended to. Reformulating is the skill that is needed to modify the programs. Programs are modified for various reasons, such as logical errors or change in specifications and requirements of the problems among others.

The last link in the chain of cognitive accomplishments is problem-solving. This is the combination of the design skills and the language features that the students use in order to apply templates learned in one system to a new system.

1.4 Cognitive abilities as factors of success

In their study, Linn & Dalbey linked the abilities to characteristices such as the general ability or intelligence of the students involved in their study as well as access to computers, previous interest in computers, gender and programming skills. They found that the programming skills of the students and their success in the final assessment were correlated with increased access to computers, interest in and previous knowledge of computers.

There was also a positive correlation between the students' general academic ability and their performance. There was no difference in performance when gender was considered. However, these results may not be applicable to today's computer science courses, in part because the research was done with programming languages that are not being taught now. Furthermore, the study done by Linn and Dalbey was done in high schools where computer science was newly introduced and the teachers did not have much experience in teaching the subject.

Both Du Boulay (1989) and Linn & Dalbey (1989) did early research projects with the objective of helping teachers become aware of the difficulties students were facing so that they could alter the courses and teaching assignments. It would be interesting to see whether these findings still hold true nowadays in our CEGEP environment. Furthermore, access to computers may not be as important an issue as it was twenty years ago. Interest in and previous knowledge of computers and computing will be more pertinent to the question.

1.5 **Cognitive characteristics**

Another type of factor considered in research done on the poor performance of novice students in introductory programming courses is cognitive style and specific abilities. Mancy & Reid (2004) focussed on what the authors referred to as working memory space (WMS) capacity and field dependency (FD). They wanted to know how useful WMS and FD can be as predictors of success in computer programming. WMS is that part of the brain that holds information temporarily, processes it, and stores it in long-term memory for further use. The capacity of the WMS is limited and information is not held for a long time. The researchers explained the concept of field dependency the following way: a learner who is not able to reorganize concepts and integrate them with past experience to solve problems is considered field-dependent. Field-independent individuals are also able to extract pertinent information and leave irrelevant materials aside.

In spring 2003, Mancy & Reid administered tests for WMS and FD to 150 first year computer science students at the University of Glasgow and correlated the findings to their performance in their assessment tasks. To test WMS, the authors used a digit span memory test based on Jacobs (1887). The memory test was not done with a computer. The students were read a series of numbers and they had to rewrite the series, forward or backward depending on the series read.

As for the FD test, they made use of a slightly modified test by Witkin et al. (1977). The students were given a set of complex figures and some simple geometric shapes to be found in the complex figures. They had to trace the outlines of the shapes. The score was the number of correct shapes found and traced with the complex figures.

From those two tests, the researchers could classify the students into categories with level of WMS and level of field-dependency respectively. The final grade of the cohort of students was calculated based on the results of four examinations given to the students throughout the course of the year. There were two practical examinations and a class test which were worth 30% of the final grade and the final examination which was worth 70% of the final mark.

Though it has been shown that WMS and FD are useful predictors of success in conceptual areas such as mathematics and statistics and that problem solvers use WMS to keep track of goals and plans and that FD students are better problem-solvers, the authors found to their surprise that WMS is not a factor of success in programming, even though WMS limitations have been shown to hinder learning progress in science education. At the same time they found that students who scored well on the field dependency test i.e. those who are considered to be field-independent scored better in their examinations.

1.6 Computer Science discipline

There have been recent attempts to define computer science education (CSE), allowing computer science to be considered as an independent academic discipline, just as mathematics or any science subject is. From this perspective, Holmboe et al. (2001) did a survey of research done in computer science education. Apart from the psychological and cognitive aspects of research done, the team noticed that most current research is being done on different ways of implementing computer technology in the teaching and learning of different subjects. They found there was insufficient CSE research done to understand the issues involved so as to improve the quality of teaching and learning in the computer science discipline, hence defining success. They tried to define what constitutes a good teaching practice so as to improve the quality of learning of programming. They suggested that a good teacher should not only know and master his or her subject matter but also that he or she must have essentially what is referred to as pedagogical content knowledge so that the student can learn to construct knowledge and succeed. According to Holmboe et al. (2001), close collaboration is needed between computer science teachers and researchers in education science, psychology and epistemology.

Students' performance is dependent on the ways disciplines are being taught and also on how the students perceive the disciplines. Programming courses are not exempted from this. Many students drop out of programming courses because they find programming difficult. They cannot process the information they receive; they do not understand concepts used in programming, the programming logic and techniques. Some teachers, knowing the strengths and weaknesses of their students, have developed and adapted ways of teaching that correspond to the learning path of the students. But they rarely document their practice. Without detailed documentation, it is difficult to deduce whether success in learning programming depends largely on the ways of teaching or on the learning methods of the students. On this question, a survey of methods to evaluate how computer science is taught was done by Carbone & Kaasboll (1998). They noticed that the most common methods to evaluate teachers and teaching methods were evaluation questionnaires, analysis of examination marks and laboratory experiments.

Similarly, many universities do a survey on students' impression of a course. The survey usually asks the level of satisfaction of the students with the course in general, as well as with the teachers and specific areas of teaching. Much can also be learned from open-ended questions which usually accompany the survey. The comments made usually target the teaching methods. Examination marks are supposed to reflect the students' understanding and competence in the subjects. The authors gave an example where students who were taught programming by means of developing formal specifications of programs improved their examination marks. But no details were mentioned concerning the student population and the reliability of the measure. However, this survey did not fully answer the question of whether student success depends on the teaching methods. It did shed some light on the delicate issues involved in learning how to program. As for experiments, not much research has been done to evaluate teaching and learning programming because setting up a controlled experiment in a laboratory is considered too complex.

1.7 Learning aspects

On the issues on learning programming, Lahtinen et al. (2005) did a descriptive and comparative correlation analysis of the difficulties students from several universities in Europe encountered in programming courses through a questionnaire which surveyed programming concepts and issues with which they had difficulties. The team also looked at the learning and teaching process as well as materials used in the learning process. They wanted to know how the students' performance was influenced by factors such as the background of the students, course contents, learning situations and materials. The learning situations that Lahtinen et al. considered were lectures, exercise sessions in small groups, practical sessions in computer rooms, studying alone and programming by themselves. Examples of learning materials considered in the survey were: textbooks, lecture notes and copies of transparencies, exercise questions and answers, and example programs. Both students and teachers participated in the survey. Fifty percent of the students surveyed had prior programming experience. Those students thought that they had a moderate to good level of programming skills.

Clearly, students in the programming courses often had different experience levels making teaching the courses challenging for the teachers. For the course contents, the opinions of both the students and the teachers on the most difficult concepts were approximately the same. Teachers and students both considered that learning by examples was the most helpful. As for the learning situations, students tended to think that they would be better off learning by themselves whereas the teachers thought that lectures were more beneficial to the students. Teachers thought that the students would also learn better with more guided exercises. Again, the results may partly be explained by the misconceptions that students have about programming. Students do not realize the amount of difficulties that they have; they sometimes overestimate their understanding. This research paper aims to help raise awareness of the misconceptions that exist in the minds of students, especially computer novices.

1.8 **Factors contributing to success**

Several researchers or teams of researchers have begun recently to look at factors contributing to success in introductory programming course. Wilson & Shrock (2001) studied factors such as previous computer experience, self-efficacy, comfort level, math background and gender among others. They tried to determine what relationship exists between these predictive factors and the mid-term course grade from 105 students they surveyed at a comprehensive Midwestern university. The students were enrolled in the first introductory programming class required in the computer science major. The authors used midterm grades as the determinant of success. They wanted to include all the students who started the course, even those who would drop out before the end of the semester. They wanted to survey as many students as possible because of the high attrition rates in introductory computer science courses.

They found out that comfort level was the best predictor of success in the course followed by math background. Comfort level was described by the authors as the ease of participation in class and laboratories, understanding the concepts in the course, and perception of the difficulties in the completion of assignments. It also involves anxiety felt while working on computer assignments.

Previous computer experience was divided into previous programming experience and previous non-programming experience. They found out that previous programming experience which included either a previous programming course or self-initiated programming plays a positive but not a significant role in student success. As for previous non-programming experience, the authors found out that game playing, one of the activities mentioned in previous non-programming experience, had a negative effect on the midterm grade.

A similar result was reported by Hagan & Markham (2000). They indicated that students who have had experience in at least one programming language perform significantly better in assessments than those with none, and that the more languages with which they had experience with, the better the performance tended to be.

In a study done in Ireland, Bergin & Reilly (2005) found that there is a strong positive relationship between programming performance and the Irish Leaving

Certificate scores in mathematics. But they also observed that there was no significant difference in performance between students with prior programming experience and non-programming experience.

Goold & Rimmer (2000) looked at some other factors affecting the performance of a group of 39 Australian undergraduates majoring in computer science in their first-year computing courses. During the first semester, two courses were given, Information technology and Basic Programming Concepts. Having successfully completed them, the students took Data Structures and Algorithms. Among the 39 respondents, only 25 of them took the Information technology course while the rest were exempted and received credits since they had some experience in the subject matter due to after school activities or pre-university courses.

Results in these three courses were analyzed. Performance was defined as the grades received for these courses. In the Information Technology units, mastery of applications software was part of the measurement of performance. It was assessed through assignments which accounted for 40% of the final grade. The same applied for the basic programming course. As for the data structures course, assessment included examination (60%), mid-term test (10%) and assignments (30%). The authors looked at how prior experience, problem-solving abilities, dislike of programming and secondary school results affected performance in the three courses.

They found the following correlations between performance and these factors: Dislike of programming was seen to have a very negative influence on performance in computer course examinations. Problem solving ability has a positive correlation with performance in the Basic Programming Concepts course, but it did not affect the overall performance in the higher level Data Structures and Algorithms course.

Those students who did some programming before university did better overall in data structures. But the results did not show any correlation between prior programming experience and examination marks. The students demonstrated better programming skills in assignments that contributed to the final grades. Secondary school results had only a slight positive correlation with success in the Basic Programming Concepts course. In the other courses, the significance was almost non-existent. Finally, gender was not a factor in performance in the advanced programming course.

1.9 Motivation as a factor of success

Bergin & Reilly (2005), in another study, also looked at the influence of motivation and comfort level on learning to program. However, this study looked at how the factors affect performance in an introductory object-oriented programming module. Object-oriented programming is a new technique of programming. Contrary to the original style of procedural programming where the focus is on the design of the processing (the design of procedures which perform appropriate actions on different types of basic data types such as integer, real or character or other types derived from the three basic ones), object-oriented programming encapsulates the data with the methods that process the data to produce a class which is considered as the data type. So whenever an object is defined as some class type, the appropriate procedures or processing are also known. Students must be acquainted with the new philosophy.

Bergin & Reilly (2005) defined motivation as the need and desire to be successful in their learning process. They also divided the students into two categories, those who were intrinsically motivated and those who were extrinsically motivated. For the latter, rewards such as grades are the motives to persevere whereas for the first group, it was personal satisfaction or personal achievements which drive them to go forward. The authors considered the students' ease with asking and answering questions in programming, and their self-esteem and self-efficacy when they described comfort level. In this study, the authors found that students who are more intrinsically motivated perform better. The higher the level of intrinsic motivation the greater is the programming success. They found that extrinsic motivation does not have a significant influence on programming performance. As for comfort level, it was found that students with higher self-esteem perform better than students with lower self-esteem, but there was no significant difference found with self-efficacy.

1.10 Time spent studying as a factor

Another factor that was studied was time spent studying. Carrington (1998) did a survey analyzing the amount of time full-time computer science students at the University of Queensland in Brisbane (Australia) spend on their homework. Students were complaining that they were overloaded with excessive work. The amount of time spent by his students in a software design course was monitored for three assignments. The time monitoring was intended to determine how students spent their time and the causes of any overload. For the first two assignments, there was a positive correlation between time spent and the marks received, whereas for the third one, it was noticed that the correlation was slightly negative. Carrington explained this result by noting that the third assignment was much more difficult and that the students spent a lot of time trying to debug their program without much success.

Although the students were told that time monitoring and reporting were an integral part of the assessment and that it was worth 5% of the mark, not every student reported the time spent on a regular basis; around 60% of students supplied data every week and 90% of them supplied the data for most weeks. In the paper it was not mentioned how many students took part in the study. It was also noticed that the reporting of time usage was concentrated during the weeks when the assignments were due. The results show that the students surveyed did not study materials related to programming regularly and that the overload reported was due to poor time management. It would be interesting to see whether Carrington's findings would also hold true for a Quebec college environment. However, it is beyond the scope of this study to repeat Carrington's work at the CEGEP level.

1.11 Learning styles as factors of success

Success in programming also depends on how students learn. A survey was done by Byrne & Lyons (2001) with 110 humanities students taking a first year programming course in Ireland. Successes in the course were defined as the ability to specify, design, code, and test a computing solution. The students were assessed on a final three-hour written examination which was worth seventy percent of the final grade and on twenty assignments worth a total of thirty percent of the final grade.

In addition to gender, prior experience, and previous academic performance in mathematics, science and languages, the authors looked at Kolb's four learning styles as factors influencing the scores attained by the students. The four learning styles are: convergers, divergers, assimilators and accommodators. Convergers are practical, preferring technical tasks and problems over those dealing in social issues. They also like to experiment with new ideas and laboratory assignments. Divergers are the opposite of convergers; instead of hands-on they prefer to observe situations from different points of view, do focus groups and work in groups. Assimilators are people who like to put information they gather in logical form in order to understand the issues; they prefer to read, go through lectures and think things through thoroughly before taking action. Accomodators are hands-on people; they like to actually try out challenging things either by themselves or in groups.

According to Byrne & Lyons, convergers perform best in all academic fields. As for gender, female students did better than male ones. But the difference was not very significant. Males had a mean score for the final examination of 39.7%

whereas the females scored 43.9%. Those with backgrounds in math and science obtained higher marks in programming exams than those having English and foreign languages as their main prior academic experience. Finally, those with some prior programming experience generally did better.

1.12 Gender dimension

There have also been studies done looking at gender differences in attitudes and perceptions in learning programming. Carter & Jenkins (1999) surveyed students from Leeds and Kent Universities in UK by giving them seven statements and asking them for their opinions. The statements used were:

1. *I find programming easy.*
2. *I prefer to work alone.*
3. *When I get stuck I will always approach a lecturer for help.*
4. *When I get stuck I prefer to ask my friends for help first.*
5. *When I get stuck I prefer to work out the answer myself.*
6. *In general, men are better than women in programming.*
7. *The lecturers are more willing to help female students than male.*

For the first six statements, no significant difference in attitude was found between males and females. But for the last statement, male students strongly believed that this was true. The authors also compared the students' performance; they found that female students performed better than male students.

1.13 Ways of learning

Furthermore, Booth (1992) did a phenomenographic qualitative study on learning to program by interviewing first time programming learners. Phenomenography is an empirical qualitative research method often used in educational research. Data is collected using interviews. These interviews are recorded and the researchers take time to analyze the data and try to understand and explain the phenomenon they are researching. The interviews are transcribed and are read and reread, in context. Excerpts of the transcript are de-contextualised, compared and grouped in different categories. In this way, researchers are able to understand the phenomenon.

Booth addressed the fundamental question of what programming means and what is demanded when learning to program. She wanted to understand how students think and how this helps them in programming. The questions she addressed in her work were the following: *What does it mean and what does it take to learn computer programming?* She found that first-year programming students learn programming in four different ways, "Coding", "Understanding and Integrating", "Problem Solving", and "Participating and Enculturation".

An extension of the study was done by Bruce et al. (2003) using the same qualitative method. They added a fifth category which they called "Following". From the interviews the authors made, we can perceive how the students new to programming learn the art. They also went further by looking at additional aspects such as the students' learning approaches and activities, their view of the programming language learned their learning motivation and their ways of seeing programming.

1.13.1 *Coding*

Students using coding focus on the syntax that makes up the language being learned. The programming language is seen as a means to develop one's competency with the syntax. They see programming as the ability to write codes. They think that the more codes one knows, the better one will be able to program.

They also spend their time looking for examples or pieces of codes in textbooks, on the internet or other sources that will help them finish their programming tasks. If they do not receive any kind of help when they ask for it, they become frustrated and disillusioned. Overall, for them, learning the syntax of the programming language is the most important part in their learning process..

1.13.2 *Understanding and Integrating*

Students using understanding and integrating consider understanding as an integral part of their learning process. They think that they would be able to write a program after understanding the concepts. Since failures at the beginning very often arise due to the fact that the students are not yet able to relate the tasks to the concepts learned, many tend to give up on programming. Those students need to persevere. They need to build their knowledge based on their schemata, i.e. prior experience and knowledge, block by block. They have to assimilate one concept before going to the next.

Their learning approach is based on understanding the concepts as well as concentrating on the task at the same time. They would write the codes only after they have understood. They would use their experience for further tasks. They would also use different sources to gain these experiences such as the internet. For them, learning programming is learning the structures and the logic of the programming language. Their motivation extends more towards understanding the big picture of programming through understanding the concepts rather than concentrating on the tasks. Lastly, for them programs consist mainly of syntax, codes, concepts and logics integrated together.

1.13.3 *Problem Solving*

Learning programming for the students using this way of learning starts with the problem. They focus simultaneously on the problem to be solved and the understanding of the concepts. They learn what it takes to solve the problems and to finish the task. They do not focus on understanding first but on the ability to end up with the solutions.

They place a high priority on planning before writing the codes. For them learning the programming language is the means to solve the problems. They are motivated by the problems they have to solve and see programming on the whole "*is about creating solutions to a problem*".

1.13.4 *Participating or Enculturation*

Students try to be part of the community of programmers, learning their culture and their ways of thinking, to gain experience and learn what programming is all about. They try to emulate their peers. They, too, need to learn the concepts and the syntax of the programming language. But they focus mainly on the communication with other programmers as their learning strategy.

They consider the programming language as part of learning the culture of programming and are motivated by the prospect of finding work in the domain. For them, programming is a culture; they can mingle with their peers sharing ideas and experience.

1.13.5 *Following*

Students in that category simply try to get through the unit. This means that they try only to complete whatever is being asked such as assignments, tests and exams. The main issue is to get marks; this informs their learning strategies and activities. Their desire to pass the course is the ultimate goal and motivation, and it is also the way they see programming.

They differ from other students in that they want the course to be structured in such a way that it matches their expectations and needs. For them, teacher feedback is crucial because they want to know whether they are on the right track and passing the course.

However, this study did not survey the relationship between success of the students in their programming courses and their ways of learning mentioned above.

2. GAPS IN THE LITERATURE

The surveys of Du Boulay (1989) and Linn & Dalbey (1989) were done at the high school level; the others discussed in the literature review were done at various universities. There has been no educational research of this kind done in our English CEGEP environment. It would be interesting to see if our college environment differs from the others. Research to answer questions about how the learning of computer programming takes place, on how students come to understand programming concepts and gain the ability to apply these concepts and on how students gain the technical and practical skills needed to be able to write efficient programs, can benefit the faculty as well as the students of computer science.

At the same time, it can help to see whether the factors such as prior academic background, hours of study, ways of studying, and regular class attendance play a significant role in the success of our English CEGEP students in their introductory programming courses. This study hopes to result in higher quality teaching and learning of computer science. Furthermore, the phenomenographic study done by Booth and Bruce et al. did not look at possible relationship between students' performance in programming and the ways they learn it. This study also attempts to see whether there is a relation between the ways of learning, Coding, Understanding and Integrating, Problem Solving, Participating or Enculturation and Following, described by Booth and Bruce et al. and the performance of our programming students. Do students using one way of learning perform or succeed better than students using another way?

3. THE RESEARCH QUESTIONS

This descriptive study set out to investigate, from a cognitive perspective, how Vanier College students in introductory programming courses learn the subject. Since there has never been a study done on the subject at the College, the study will focus on several factors that contribute to the success in the courses. Can the ways of learning programming of Vanier students can be categorized according to Booth (1992) and Bruce et al. (2002)? Is it possible to compare how successful the students in each category are? Success in this context means passing their programming courses.

From the teaching experience accumulated throughout the years spent in teaching computer science and programming, from numerous discussions with colleagues in the Computer Science department and from anecdotal evidence, the following hypotheses were initially selected to be tested:

- *Computer Science students using understanding and integrating succeed better than students using following, coding, or problem solving.*
- *Students using problem solving succeed better than those who use participating and enculturation.*
- *Students who use coding perform better than those who prefer participating and enculturation.*

In addition, the author set out to see whether there is a gender difference in how students learn programming. For example, it is interesting to know the most common way of learning (described by Booth and Bruce et al.) of female students compared to male students who are taking programming courses.

CHAPTER 3

RESEARCH DESIGN

I. METHODOLOGY

In order to answer the initial research questions posed, data were gathered in two different ways, a survey questionnaire (Appendix A) filled by the participants and a semi-structured interview. The survey questionnaire was used to collect quantitative data to be analyzed using the statistical package SPSS for Windows. The interview was used to corroborate the answers given by the students in the questionnaire.

One question in the questionnaire explicitly asked the participating students to choose a way of learning which they feel is the most appropriate for them. The question asked was the following:

> *Which of the following describes you the best, when you learn programming? (Choose only one.)*
>
> *a. I learn the syntax of the programming language first and then spend a lot of time at the computer testing and running programs.*
>
> *b. I need to understand the concepts before I can apply them to practical tasks. I need to understand a concept fully before learning others.*
>
> *c. I start by analyzing a problem and then look at the concepts and syntax necessary to solve it.*

d. *I gain experience and learn what programming is all about by learning the cultures and the ways of thinking of experienced programmers and try to follow their example.*
e. *I learn by trying to do all the assessment tasks that are part of the course requirements.*
f. *I do not know.*

The question placed the students into the categories enabling the researcher to test the initial hypotheses.

There was a set of open-ended questions at the end of the questionnaire (Appendix A). Learning how to program is best done by doing programming assignments, by practising coding. Hence, how the students approach a programming task can give us a fair amount of information on how they learn. We can thus categorize the students by the ways they deal with the difficult task of learning the art of programming. The open-ended questions posed in the questionnaire were:

(1) Can you elaborate on how you do a programming assignment?
(2) Is there a set of steps that you usually take to complete it? Please explain.

A pre-test of the survey questionnaire was administered to 33 first and second year computer science students in order to detect any flaws or ambiguities. Appropriate modifications were made resulting in the final questionnaire used in the current survey. In the pre-test questionnaire, students were asked to choose from a range of numbers their average high school grade, their final grade in their high school mathematics course, the number of hours they work outside school, and the number of hours they spend studying. Accuracy was lost when medians were calculated for the correlation analyses. Thus, in the revised questionnaire, students were asked to enter a real number for each of these variables. Also, a set of questions

(Questions 23 through 30 in Appendix A) about the activities describing their styles of learning were added. These questions were used for triangulation purposes.

2. DATA COLLECTION

The researcher went to class and explained to the students the purpose of the research. The students were then asked to fill the questionnaire. The quantitative data collected were analyzed by the statistical program SPSS. The open-ended questions at the end of the questionnaire were analyzed using content-analysis methods. The semi-structured interview was done one to two weeks after the questionnaire. Only students who volunteered were interviewed. The interviews were mostly done on an individual basis in the office of the researcher except for two pairs of students who asked to be interviewed together. The students were reminded of the purpose and importance of the research and that their answers would remain confidential. The interviews were taped. Each interview lasted around 15 to 20 minutes.

3. PARTICIPANTS

Fifty-eight Vanier College students (other than the ones who did the pre-test), thirty-four (approximately 59%) of whom being in the Computer Science Technology Program and twenty-four (41%) of whom being in the pre-university Science Program, volunteered to participate in the survey. The Computer Science students were in the first and second semester. The first semester students were taking the introductory programming course at the time of the survey which was conducted in March 2007. The second semester students had already taken the course and received a final grade for it. As for the Science students, they were taking the complementary programming course which is equivalent to the introductory programming course taken by the Technology students. Even though Computer Science students were using the Java programming language and Science students were using the C++ programming language, they all had the same competencies, objectives and standards to fulfill.

The percentage of female students taking the course was low; only 8 out of the 58 participants were females. This means that looking at the difference in gender performance did not yield statistically significant results.

Both computer science and science students had their theory classes conducted in an ordinary classroom or lecture room and their practical classes in the computer laboratories. In the theory classes, the students were mainly lectured to; programming concepts were being taught using examples. Sometimes the teacher would demonstrate, in the classroom, some of the examples using a laptop connected to a video projector.

During laboratory sessions, each student had a computer to himself/herself to test his/her programs since the labs are mostly hands-on. There were more practical or laboratory hours than theory classes in a ratio of almost two to one meaning that the students had a fair amount of one to one contact hours with the teachers. On top of that, the students are usually free to come to any available computer laboratories when they want to work during their free periods. They have plenty of opportunities to practice their programming skills.

4. QUANTITATIVE DATA

Data were collected using quantitative as well as qualitative methods. For the quantitative data, the instrument used was the questionnaire developed by the researcher especially designed for the survey. It collected data on the following main items: (a) gender, (b) high school average grade, (c) high school prerequisite mathematics grade needed to enter the Computer Science Program, (d) previous computer knowledge, (e) prior programming experience, (f) class attendance, (g) ways of learning and tackling assignments, (h) number of hours worked outside the curriculum, (i) hours spent studying and doing the course assignments, (j) the level of comfort in understanding the concepts, (k) the level of comfort in problem solving and (l) the level of comfort in doing the course assignments. The three questions on the last three factors depicting corresponding levels of comfort used a Likert scale from 0 to 4, corresponding to (0) strongly disagree, (1) somewhat disagree, (2) somewhat agree, (3) strongly disagree, and (4) don't know. Questions (14) to (16) in the questionnaire (Appendix A) given as statements were as follows:

(14) I have difficulties in understanding the concepts of programming.
(15) I have difficulties in problem solving.
(16) I find computer science difficult.

These variables were correlated with the final grade received for the introductory course. The final grade for the introductory programming course for the computer science students was calculated in the following way: assignments and quizzes were worth 40% of the final grade; three tests were given worth 15%, 20% and 25% respectively. As for the course for the science students, the assignments and exercises accounted for 25% of the final grade; the three tests had a value of 20%, 25% and 30% respectively of the final grade. The final grades received by the respondents were officially handled by the coordinator of the Computer Science department who oversaw the smooth running of the courses given by the department.

Frequency tables (Appendix D) concerning the following variables, gender, access to a computer at home, mother tongue, high school language of study, prior computer knowledge, prior basic programming, use of the recommended text, working in group doing an assignment, working in group preparing a test, participation in class, missing classes and ways of learning were generated.

Other variables involved in the survey relate to the following:

(a) understanding programming concepts before using them. (b) trying examples to understand the concepts of coding, (c) learning syntax before applying it to solve problem, (d) learning by analyzing and testing working programs, (e) analyzing problem and write algorithm before coding, (f) doing assignment by discussing with peers in group, (i) copying or emulating more experienced programmers and (j) simply doing all assessment tasks.

These variables described the learning process of the students. The questions posed as statements also used a Likert scale ranging from (0) to (4) corresponding to strongly disagree, somewhat disagree, somewhat agree, strongly agree and don't know respectively. Examples of the questions (see Appendix A) are:

(23) I try to understand the programming concepts before using them.
(25) I learn the syntax of the programming language before applying it to solve a problem.
(26) I learn programming by analyzing and testing working programs.
(27) I analyze problems and write the algorithms before attempting to do some coding.
(29) I learn programming by copying and emulating others who are more experienced.

These questions are similar to the way's of learning; they are given as a triangulation method. Correlation analyses were generated to look at relationships between the variables mentioned above and the final grade of the students. Independent-samples t-tests were conducted to test the hypotheses.

5. QUALITATIVE DATA

Additional qualitative data was collected using a semi-structured interview. Ten students among the Computer Science Students volunteered to be interviewed on the basis of their answers to the questionnaire. The following questions were used during the interviews. These questions address the question of how students go about learning to program:

1. Tell me why you chose computer science.
2. What is a programming language to you?
3. What are the main techniques that you use when you learn to program?
4. What type of assessment tasks help you most in learning to program?
5. Can you describe how you go about writing a program?
6. How do you overcome the frustration when your program is not working?
7. Is there anything that you would like to see improved in the ways learning how to program is being taught?

The qualitative data collection was done as a triangulation method and expands on the answers given by the participants in the survey questionnaire. The students were asked to express whatever views they have about their learning process in programming. The interviews were recorded and transcribed. The answers were then analyzed. The procedures for the interview are given in Appendix B.

6. ETHICAL ISSUES

All the participants took part in the survey voluntarily; they signed a consent form (Appendix C). Those who were under 18 had obtained their parents' permission to participate. They were given an explanation of the purpose of the research project and why their participation was helpful. They were explained the process by which the research would be conducted, how the data would be used and to whom the results would be reported. In doing so, any deception could be avoided. Furthermore, participants were told that they could withdraw from the research for any reason whatsoever without being penalized in their academic performance.

Moreover, the participants were informed that their personal data would be protected and would remain confidential. The identifying descriptions of the data would be removed so that their privacy is protected.

CHAPTER 4

DATA ANALYSIS AND RESULTS

1. QUANTITATIVE DATA ANALYSIS RESULTS

Vanier College is a college where the language of education is English. But it is interesting to note that the student population is rather particular. This particularity is also reflected on the group of respondents to this survey. The frequency tables (Appendix D) show that less than half (43.1%) of the students have English as their mother tongue. Exactly half of the students had done their high school studies in English.

Furthermore, 53.9% of the respondents said that they had a good or very good knowledge of how computers work, 27.6% said that they had a fair knowledge and the rest had a rather poor or no knowledge at all. These students who had not done any programming before starting the introductory course accounted to 48.3% of the respondents. Among the rest, 48.2% said that they had very little or a little prior experience in programming. Only two of the students firmly stated that they had done much programming before. When asked whether they usually used the recommended textbook, 49.9% replied that they used it sometimes, or often, or all the time. The rest said that they never or rarely did.

It is also interesting to note that more than half (53.7%) had at one time or another worked in group when doing their programming assignments. But, when they were asked whether they prepared for tests together, twenty five (43.1%) students said that they never studied for tests with others whilst 16 students (27.6%)

said that they sometimes or often worked together. The majority (74.2%) of the students never or rarely missed classes.

When asked about the level of comfort with programming, 63.8% answered that they did not have difficulties understanding the programming concepts; 74.1% strongly disagree or somewhat disagree that they had difficulties solving problems and 69% replied that they did not find computer science difficult. One thing that they had in common is that they all had access to a computer at home.

The respondents were also asked explicitly in what category they would place themselves among the five ways of learning; they were asked to choose only one, that which would fit them the most. The following statements were used in the survey question to describe the different ways of learning:

1. I learn the syntax of the programming language first and then spend a lot of time at the computer testing and running programs (Coding)
2. I need to understand the concepts before I can apply them to practical tasks. I need to understand a concept fully before learning others. (Understanding and Integrating)
3. I start by analyzing a problem and then look at the concepts and syntax necessary to solve it (Problem Solving)
4. I gain experience and learn what programming is all about by learning the cultures and the ways of experienced programmers and try to follow their example (Participation and Enculturation)
5. I learn by trying to do all the assessment tasks that are put for the course requirement (Following)

The following table shows the result:

Table 1
Frequency on ways of learning

Way of learning	Number of responses	Response rate
Coding	14	24.1%
Understanding and Integrating	13	22.4%
Problem Solving	16	27.6%
Participation and Enculturation	3	5.2%
Following	6	10.3%
I do not know	5	8.6%
Blank	1	1.7%

Table 1 shows that very few respondents were in the categories of "Participation and Enculturation" and "Following". Almost the same numbers of respondents fell into the other three categories, "Coding", "Understanding and Integrating" and "Problem Solving". Only three students said that they fell into the "Participation and Enculturation" category and only six respondents said that they learned programming by doing all the assessment tasks that were given.

. Since the number of students in these two categories is insignificant, these results led to a revision of the initial objectives of the project. The hypotheses stating that "Students using problem solving succeed better than those who use participating and enculturation" and "Students who use coding perform better than those who prefer participating and enculturation" could not be tested since significant results would not be achieved due to the small number of students favouring "participating and enculturation".

Furthermore, since "Following" also had few respondents, instead of the hypotheses "Computer Science students using understanding and integrating succeed better than students using following, coding, or problem solving", the following hypotheses "Computer Science students using understanding and integrating succeed better than students using coding or problem solving" was tested.

2. CORRELATION ANALYSIS

Since the number of respondents was not large, Spearman Rho correlation and Kendall's tau correlation tables were generated for several variables. There is a general belief based largely on anecdotal evidence, that a student with a higher high school average and a high school mathematics grade will be more successful in programming. This may hold true for other institutions but at Vanier from the responses that were collected the result was not quite what was expected.

The high school average and the high school math grade do have a slight positive effect of the final mark that the students received but the relation was not significant. However, we did notice that there was a significant positive relationship between high school average and high school math grade and the midterm mark.

These results may be explained by the fact that the students in the programs have been selected and accepted because they had satisfied the requirements that are necessary and they had the prerequisites to be in their respective programs. They just started college studies and they were very motivated. This explains their midterm results; the higher the high school average and the higher their high school math grade the higher their midterm marks were. After some time spent in the college, the students presumably started to assimilate to the college culture. Table 2 below shows the result of the Spearman's rho correlation addressing the relationships discussed above.

Table 2

Spearman' rho correlations among High school average, High school math grade, Midterm mark, Final mark and Ways of learning

	High school average	High school math grade	Midterm mark	Final mark	Ways of learning
High school average	1.00				
High school math grade	.60**	1.00			
Midterm mark	.44**	.35*	1.00		
Final mark	.20	.10	.29*	1.00	
Ways of learning	-.14	-.01	.20	.10	1.00

** $p < .001$

* $p < .005$

Moreover, Spearman Rho and Kendall's tau correlation analysis show that there is a significant negative relationship between prior computer knowledge and prior basic programming experience with difficulties understanding concepts, difficulties solving problems and finding computer science difficult (as shown in the Spearman's rho Table 3). The latter variables result in a decrease in the final mark.

Thus, prior computer knowledge and prior basic programming knowledge play a positive role in success in learning how to program. Having less prior computer knowledge and less prior basic programming experience leads students to have lower final marks.

Table 3

Spearman' rho correlations among Prior computer knowledge, prior basic programming, Difficulties understanding concepts, Difficulties solving problems, and Find computer science difficult

	Prior computer knowledge	Prior basic programming	Difficulties understanding concepts	Difficulties solving problems	Find computer science difficult
Prior computer knowledge	1.0				
Prior basic programming	.28*	1.0			
Difficulties understanding concepts	-.28	-.29*	1.0		
Difficulties solving problems	-.31*	-.43**	.68**	1.0	
Find computer science difficult	-.35**	-.23	.66**	.60**	1.0

** $p < .001$ * $p < .005$

A correlation analysis was done between time spent studying programming (M = 4.97, SD = 6.80) and the final mark (M = 74.22, SD = 14.31). Spearman's rho correlations did not show any significant correlation between these two variables (r (53) = .03). The same result was obtained with Kendall's tau_b. The result came as a surprise. This might be explained by the fact that the students were asked explicitly in the questionnaire how many hours they thought they spent studying the programming course. At that particular moment, they did not have a good idea of the amount of time they spent in each course. A more appropriate method can involve asking the students to log on and keep track of the time they spend on programming.

Due to the multicultural and multiethnic college population, there are many students who do not have English as their main language. A Spearman's rho correlation addressed the relationship between Mother tongue (M = 1.57, SD = .50) and Difficulties understanding concepts (*M* = 1.28, *SD* = .87). Similarly, the correlation addressed the relationship between Mother tongue (*M* = 1.57, *SD* = 5.0) and Difficulties solving problems (*M* = 1.1, *SD* = .88) and between Mother tongue (*M* = 1.57, *SD* = 5.0) and Find computer science difficult (*M* = 1.2, *SD* = 1.05). Table 4 shows that there is a slight significant correlation between mother tongue and difficulties understanding concepts (*r* (56) = .27, p < .004 for an alpha level of .01) as well as difficulties solving problems (*r* (55) = .29, p < .003 for an alpha level of .01) and a significant correlation between Mother tongue and Finding computer science difficult (*r* (56) = .35, p < .001 for an alpha level of .05). This means that those students whose mother tongue is not English were having more difficulties in their studies.

Hence, factors that are of importance and that may promote the success of our students in the introductory programming course are prior knowledge of computers and prior basic programming experience. The more experience the students have, the less difficulties they have in understanding the concepts of programming and in problem solving. Thus, they will find computer science less difficult. Language also plays a vital role in the field of programming. English speaking students have fewer difficulties in their learning process.

Table 4
Spearman' rho correlations among Mother Tongue, Difficulties understanding concepts, Difficulties solving problems and Find computer science difficult

	Mother tongue	Difficulties understanding concepts	Difficulties solving problems	Find computer science difficult
Mother tongue	1.0			
Difficulties understanding concepts	.27*	1.0		
Difficulties solving problems	.29*	.68**	1.0	
Find computer science difficult	.35*	.66**	.60**	1.0

**$p < .001$ *$p < .005$

Lastly, it was interesting to observe that there was no significant correlation between the ways of learning and the final mark obtained by the students. This might mean that the hypotheses put forward in the research study would not hold. Then, it would make more sense to see whether there was a difference between the final marks received by the students who use coding and understanding and integrating, coding and problem solving and eventually understanding and integrating and problem solving. Thus, independent t-tests were used. An independent-samples t-test was conducted to compare the final mark in programming received by students using coding and.students using understanding & integrating. The t-test showed that there is no statistically significant difference between the final marks in programming for students using coding (M = 71.4, SD = 16.5) and students using understanding and integrating (M = 71.6, SD = 11.1) for (t = -0.034, p = .973). Similarly, the t-test showed that there is no statistically significant difference between the final marks for students using coding (M = 71.4, SD = 16.5) and problem solving (M = 81.2, SD = 12.6) for (t = -1.835, p = .077). Lastly, t-test revealed that there is no statistically significant difference between the final marks for students using understanding & integrating (M = 71.6, SD = 11.1) and problem solving (M = 81:2, SD = 12.6) for (t = -2.145, p = .041).

Therefore, we cannot deduce that Vanier College programming students using coding succeed better than those using understanding and integrating. The same conclusion can be said about students using coding and problem solving. Similarly, it cannot be said that students using understanding & integrating are more successful than those using problem solving.

3. QUALITATIVE DATA ANALYSIS

The qualitative data analysis aimed to examine whether students at Vanier are similar to or different from students of other institutions in their ways of learning how to program according to the categories of Booth (1992) and Bruce et al. (2003). The data analyzed came from the open-ended questions in the survey and semi-structured interviews with 10 students from the Computer Science program at Vanier. The initial qualitative analysis involved a thorough process of reading, re-reading the answers and extracting pertinent keywords and parts of texts that defined the action of learning to program. A list of the pertinent chunks or expressions was drawn. The latter were then grouped together by categories. This process was done for both the open-ended questions and the transcripts from the interviews.

4. OPEN-ENDED QUESTIONS

For the open-ended questions from the survey questionnaire, six categories of answers could be drawn. The first four correspond to Booth's (1992) ways of learning. They were Coding, Understanding & Integrating, Problem Solving, Participation & Enculturation, Examples and Miscellaneous. The table in Appendix E shows the final grouping. It shows that Vanier College students are no different than students in other institutions when it comes to the ways they learn computer programming. The most common ways are still Coding, Understanding & Integrating, and Problem Solving. There are a still a few who look to their more experienced peers to help them complete their work. Also, when doing an assignment, there are some students who learned by examples. Four respondents answered that they were inspired by other programs or used older assignments as guidelines to try to finish their assignment. Finally, in the answers for the open-ended questions, five students did not elaborate on how they proceeded when they had to do a programming assignment.

5. INTERVIEWS

Ten students volunteered to be interviewed. Among the 10 volunteer students, only two were female. The purpose of the interviews was to corroborate and to validate the answers given by the respondents to the questions of the survey questionnaire and at the same time, to better understand how Vanier College students learn computer programming. The answers given during the interviews also shed light on how students come to understand concepts and gain the ability to apply them and why failure rates among programming students are high.

The transcripts of the interviews were analyzed by multiple iterations. All were novices to programming; they had their first encounter with computer programming when they started their college program. The one who was not a novice had a "*limited experience with Pascal*", a programming language which is not being taught at the college any more. The programming language Java was the language that all the interviewees were taught for their introductory course. This corresponds to the survey results which show that only two respondents had done much programming before and those who had some prior programming experience had only a very limited one.

The second point that was considered concerned the motivation of the students towards programming and towards starting a program that was considered to be difficult. Although in the quantitative survey, no question was asked concerning the motives of the students to take computer science courses, it was interesting to examine the responses to the question posed to the interviewees asking why they chose the Computer Science program. The overall motive is that they are simply interested in Computer Science, but for different reasons. One became attracted to the topic because of a report she had to produce in her workplace when she had to ask a programmer for support. Another student said that it would be interesting and attractive to get a job in the field. Challenge was the motive for one respondent whereas for another who likes to play games the possibility to write his own computer games was the reason why he entered the program. The others were interested because they had been around computers or had been fascinated by them since an early age. One of them said that he hoped that in doing Computer Science, he would know how computers work.

Three other observations were made from the interviews; they described the learning process itself, the difficulties the students encountered and the ways the students coped with the difficulties encountered during the period. It was pertinent to look at the latter because it formed part of the process of learning to program. The observations are described in the following paragraphs named Learning process, Difficulties encountered and Methods of coping with difficulties.

6. LEARNING PROCESS

The learning of programming usually is not a trivial activity; there is always a roadblock along the way. This is because there are so many new concepts and items to learn before one can say that one has assimilated enough in order to write a decent running program. The students use different learning approaches and activities. A student may use a mixture of some of these approaches simultaneously. This is usually the case. Those approaches were extracted from the transcripts from the interviews and listed. They were then grouped into categories namely, Syntax, Concepts, Analyze, Peers, Examples and Others (Appendix E). From these categories, the learning process was observed. Following are comments and excerpts from the interviews concerning the various activities in the students' learning process.

6.1 Attending lectures

Most of the students interviewed attended the lectures in order to understand the concepts. Some of them complement their learning of the concepts by reading the book and doing small assignments. Here are some excerpts of what some of the interviewees said about attending lectures:

".. I learned by, you know, being at the lecture, listening to the lectures, listening to what the teacher is saying ..."

".. You come to class, you have to listen to the teacher because the teacher knows what he is talking about. The second thing is to read, read the book. ... The third thing is to do the assignments. ... So the thing is to write program, to do the exercises in the book, you know, something like that."

".. In the book, there are two parts, the part ... the theory part explains each program, how it's going to do and the part with exercises. In the book, I learn the theory part and I practise the assignments from the book"

".. I go; I attend all the classes. It's supposed to work and I take notes."

".. We have class time two times a week and we also have lab time two times a week. The way it works is that we learn stuff we go over. We have a book; it's very good too. The teacher goes over stuff in class, ... And then we go to the lab and we actually practise it."

Some students have to write codes and run the codes in order to understand. But they still rely on the notes from the lectures which they attend. This is reflected in the following:

> *".. For one reason, when taking classes, they explain there ..., it helps to give me a general idea of what I am supposed to be doing. But once again, when I get to the computer, there is where the real world integration takes place. After writing the code and seeing the result of each action really. That's really when I start to understand and from the beginning to even now any new concept."*

> *".. It's kind of gibberish, like the concepts, it's really hard to understand until I actually wrote some codes and did some little programs in class, some applications."*

> *".. What I learn is that, if you actually write or type something, you will understand and memorize it more."*

6.2 Learning by examples

Apart from the lectures in class, students learn by examples. They rely on the examples given in class and in the textbook to help them learn the programming concepts.

> *".. I learned a lot doing the exercises, the assignments that were given."*

> *"..Going home, I read the book, read the chapter and then I read the examples in the book, code them in the computer, and see how it works and so on."*

".. I try to do some of the book examples and then try the assignment or homework."

".. I go through the book and do the examples. I try to apply them but ... I do some few examples here and there. If you actually do all the programming you actually become very knowledgeable."

In this last comment, by "*do all the programming*" the student must have meant "do all the examples".

6.3 Applying and Integrating

Furthermore, for most of the students, assignments are important in their learning process. Some prefer short assignments where they can put into practice each concept one at a time whereas others prefer long assignments where they can apply and integrate what is being learned in a real working program. They said:

" .. Assignments are good to keep coding and then to teach students to really trace the codes also."

" .. So if we are actually taking short assignments, five rather than one big assignment, everyone will understand; focus on understanding and then, at the end, we can just be given a project where we have to add it all together and make one thing."

" .. I learn just by doing the assignments. I am just doing like studying; I just do what have to be done. This is how I learn the syntax; but sometimes when I am curious about something, I test it out to see how it works."

" .. And if you do his assignments in class, then you master everything."

All the students interviewed used a combination of different activities to learn programming. Those activities can be grouped into four categories, Lectures/ concepts, Coding, Examples and Assignments. Other practices could not be grouped because they are isolated. That is why they were placed in the category called Miscellaneous. The most striking among the ways of learning was the fact that some students said that they rely on the teacher to pass the knowledge to them but they did not elaborate on how they wished to be taught.

7. DIFFICULTIES ENCOUNTERED

Since programming is not considered an easy task, students are always encountering a roadblock, difficulties that can deter them from continuing or slow down their progress. Coping with the difficulties helps in the learning process. Difficulties arose when the program the students were writing and testing did not work or did not return the anticipated results.

The tables in Appendix F and Appendix G illustrate the difficulties encountered and how they are being overcome respectively.

7.1 Syntax errors

Most often errors occur when the students do not use the right syntax or omit to use the corresponding functions. These errors are detected when the program is being compiled and are easily corrected. They made the following statements:

".. Start with errors in compiling. Try to compile ... missing a semi-colon here, missing a semi-colon there and so on and so forth."

" .. I just find what's wrong with it looking what's wrong with it and then one by one fix all the errors."

" .. I forgot to import such and such class."

Students learn by not making the same mistakes again. Sometimes, it makes sense to look over the program again and trace it on paper to see whether it is doing what it is supposed to do. One interviewee had it the following way:

".. I understand the program even more once it's finished and I look over it and I trace it on paper and I trace what's going on and I kind of say, well I kind of realize, ok, that' exactly how it works."

7.2 Difficulties in problem-solving

Novice programmers commonly have difficulties in problem solving, which is a prerequisite to succeeding in programming courses. Among the ten interviewees, five of them mentioned that they had difficulties in problem solving.

".. The problem that I have right now is to find what is the problem,"

" .. I guess we did not know the logics and how to proceed with the problems."

".. if you want to really program you need to step away from the book, you need to step away from the computer, you need to really think what's the problem and how do you plan of solving it. You know what the problem is and think of the solution. ... You just can't sit there and expect the solution to come to you."

7.3 Difficulties understanding the concepts

The difficulties of not being able to solve problems are mainly related to the fact that some students have a hard time understanding the concepts of programming. By concepts, it means how programs are structured and what the main characteristics of different tools and modules in the language used.

" .. But I have to say that first semester I still did not understand the concepts that well, you know. Second semester started being more clear as to what's this whole ... you know ... What is a class? What's object-oriented programming, you know?"

" .. That's the problem. Some people, now in programming are actually understanding the concepts that we are supposed to understand in the first semester now. It's really late."

" .. and the concepts that we still don't understand from last semester; now we are stuck; we are stalled."

8. COPING WITH DIFFICULTIES

8.1 Perseverance

Different students have different ways of coping with the difficulties. It is in dealing with those difficulties and overcoming them that they learn. The most common way of solving a difficult programming problem is perseverance. The students tried to see what the main causes for their errors were and to find different sets of solutions. They very often found their own way out. Sometimes they would turn to their teachers or classmates for guidance.

".. If something is not working, I usually, I just keep trying over and over again. I am in front of the computer, sometimes ... hours later, I'll get it. ... Sometimes I just stop and go and ask my teacher where I am doing wrong. I have done that sometimes but I am some sort of stubborn, I would want to learn it on my own because once I've done it on my own, it helps me learn more than if someone tells me."

8.2 Seeking help to overcome difficulties

Students most rely on other sources when they cannot continue advancing in their programming assignments. Some turn to the books to look for similar examples that can help them. Some turn to the teacher since they consider that it is the right way to do.

".. when the program is not working, it' best to ask questions to the teacher. That' the best ... Because if you don't ask, you don't ... you can't solve the problem."

Some most often turn to their peers because they feel embarrassed to ask the teachers because they feel that they are supposed to know the answer to the questions that they had to pose.

" .. What we do is either we get help from other students who actually know what they are doing. ... It' a little embarrassing to ask questions that you are supposed to know. ... You are supposed to know the answer."

" .. Most of the teachers ... They just give us the materials and tell us to do. They don't do much. They tell us to work; they tell us we are worthless, something like that."

8.3 Relaxation as a mean to cope with difficulties

Others just take a break from programming and later come back to the problem thinking that a clearer mind would help them.

> " .. *Most of the time, what I do is, I take a break. I walk away from the program that I cannot solve and I think of it again, go back to the first step I see if I made an error in the logic and see if I can break down the logic even more, ... And if that does not work, then I call a friend.*"

> " .. *I think I just take a break from it, do something else for a while. Then come back and it might become a bit more clear.*"

Attempts to correct errors and overcoming difficulties are usual activities in computer programming. Whatever approach a student used would help in the learning process.

From these interviews, different ways of learning programming can be observed. Different students use different approaches in order to achieve the objective writing a working program that will solve a problem. They should be able to apply whatever techniques they learn to be able to produce systems that satisfy the specifications and requirements asked by the user of the system.

CHAPTER 5

CONCLUSION AND DISCUSSION

This study answered questions on how Vanier College computer science students learn programming. The original question as to whether their ways of learning programming can be categorized according to Booth (1992) and Bruce et al. (2002) was answered. But, results from the analysis of the quantitative data did not support the initial research hypothesis. Few students preferred participating and enculturation or following. At the same time, the hypotheses regarding whether students using one way are more successful than others using another way were examined but could not be proven true. Simultaneously, the survey looked at the factors that influenced their success in programming.

1. CONCLUSION

1.1 Ways of learning

From the quantitative survey conducted within students taking programming courses, three ways of learning programming were observed to be the most popular among the students. They are coding, understanding & integrating and problem solving. Among the students surveyed, some students focus on the syntax of the programming language to write the codes.

Almost the same number of students prefers to understand the concepts of programming before attempting to write a program. The third most important way of learning by the students uses the students' abilities to solve problems. By simply solving the problems, these students learn the concepts of programming and at the same time the syntax of the language.

The other two ways put forward by Booth (1992) and by Bruce et al. (2003), namely, participating and enculturation and following are applied only by a few of the students. In this study, since correlation analysis did not show any significant relationship between the ways of learning and the final mark received by the students it cannot be shown which way of the first three ways has been of more help in the successful completion of their first introductory programming course.

It cannot be said that a student using coding would succeed better that one using understanding & integrating or a student using coding would be more successful that one using problem solving. Similarly, it cannot be shown that problem solving is a more appropriate way to use than understanding & integrating to pass the course. Independent-samples t-tests were used to confirm the findings.

Using the t-tests, it was shown that there was no significant difference among the three ways taken two at a time. After conducting the t-tests it was found that students have the same chance of success regardless which of the three ways of learning they use.

4.2 Factors influencing success

The quantitative survey also looked at some factors that might help encourage the success of the students in the introductory programming course. Although anecdotal evidence suggests that high school average and high school mathematics contribute greatly to the success in introductory programming, this was not confirmed in the survey done with the Vanier College programming students. The relation between the final grade and the two independent variables was not significant.

On the other hand, prior computer knowledge and prior basic programming knowledge positively influence success. This was shown in the correlation analysis: those who did not have prior computer knowledge or prior programming knowledge were more prone to have difficulties in understanding the concepts, difficulties in solving problems, and to find computer science to be a difficult subject. This in turn leads students to be more inclined to fail the course. This study also indicated that English speaking students have fewer difficulties in their learning process. However, significant correlation between mother tongue and success was not established. This may be explained by the fact that some students whose mother tongue is not English are very fluent in English because they had their high school education in an English environment.

1.3 Triangulation

The open-ended survey question and semi-structured interview were used in order to support the findings of the quantitative survey especially regarding the ways the students learn programming. It was clear from the responses in the interviews that the ways of learning correspond to the results of the survey. The content of the interviews was analyzed in terms of three angles, the learning process, the difficulties students encountered during the learning and the ways the students cope with the difficulties.

It was observed that the three preferred ways of learning programming discussed in the quantitative analysis hold true. The students interviewed used codes to learn the syntax of the programming language in order to do their programming assignments. It was also observed that students generally attempt to understand the concepts by going to the lectures and reading the book and attempt to write the programs asked of them.

Some of the interviewees explicitly talked about problem solving as a means to learn programming. These are the three ways of learning most students use in their learning process. Very little mention was made concerning reliance on fellow students to teach them how to program. There was no mention of enculturation where programming students tend to mix with other peers to learn their ways of thinking.

The same phenomenon was observed when the students talked about how they resolved the difficulties encountered during programming. The students persevered by going back to the lecture notes and their books to look at the syntax and the concepts. They tried to solve their problems on their own at the beginning before going to consult with the teacher or ask their friends. The teacher still plays an important role in learning; the teacher is the last resort person when things really do not work, when the students really cannot see the solution to their difficulties.

" .. Our teacher will help us out as much as he can because he wants us to understand."

" .. If I don't do all the three things, in the assignments I would be stuck and I need the teacher."

1.4 Motivation and expectation

Besides the ways of learning discussed, the motivation to study computer science was discussed. Most of them said that they were interested in computers at a very early age. They wanted to know more about how the whole computer system works. Others considered a career in computer science was the motivation behind their choice. One became interested because he was completing a report that needed the use of computers and computer knowledge.

During the interviews it became clear that success in programming also depends on the way computer programming is taught. Some students prefer that teachers introduce the concepts during the lecture sessions and do the demonstration during the laboratory periods. Others prefer that the whole notion of programming is demonstrated throughout the lectures using visual tools in class.

> *" .. We learn the concept and in class he also actually goes over the application of it. He does not just show what it means. He actually shows us examples; he actually has a computer hooked up and he actually does program examples in the class."*

All of them appreciate that they can practice the programming during the laboratory sessions.

The interviews revealed that there is some rote learning taking place. Some students tried to learn the concepts and the syntax of the programming language by heart without trying to understand how to apply them. They just learned the situations where they can apply the concepts and the rules.

> *" .. What I learn is that, if you actually write or type something, you will understand and memorize it more."*

> *".. Like you say, you just read the book just before the exam."*

Rote learning may be counterproductive in their learning process. If they come across new situations or new problems that they have never seen before, they will be disoriented and they will not be able to solve them. This problem is interesting and further research may be needed to see whether this phenomenon is widespread among computer science students.

2. DISCUSSION

Overall, Vanier College programming students mostly learn programming in three ways: coding, understanding & integrating and problem solving. They use the textbook as a guide and rely on the teacher as a last resort in order to solve their difficulties. From the interviews, it appears the factor that plays the most important role in their successful completion of the introductory programming course is perseverance.

However, quantitative data showed that there is no correlation between number of hours of study computer science materials and the final marks. This discrepancy is interesting; several questions can be asked. During their study periods, what activities are they performing? Are they at ease when they are doing their assignments or are they struggling to correct their errors spending a lot of time debugging? How much time do they generally spend for their homework? Do they need more time to assimilate the concepts because they are reluctant to ask their teachers for help?

Further research is needed surveying the students' activities and how they spend their time on these activities. Also, the quantitative data show that high school average and high school mathematics only play a slight role in the success for the respondents of the survey.

Since it has been shown by independent-samples t-tests that success in introductory programming does not depend on either the three ways of learning, it will be interesting to see how Kolb's four learning styles described by Byrne & Lyons (2001) will influence the performance of Vanier College's computer science students. Can the students be categorized into convergers, divergers, assimilators and accommodators? Will there be a difference in performance among them?

3. LIMITATIONS OF THE STUDY

Fifty-eight Vanier College students participated in this study. The participants included first, second and third semester computer science students as well as science students taking the complementary programming course for science. For the interviews, only Computer Science students volunteered to come forward; there was not a single science student who agreed to come and talk to the researcher.

The student population of Vanier College is unique. Only 40% of the students have English as their mother tongue; 20% have French as their mother tongue. The rest is made up of students whose mother tongue is neither English nor French. This was reflected in the students in this study. It was noticed that some language difficulties arose in the answering of the questionnaire and also during the interviews. Many students do not understand computer science jargon. They really are mixed up discussing programming terms, syntax, concepts, and logic even though these terminologies are being used in their classroom, course materials and lectures. Some think that they mean the same thing. Furthermore, during interviews some students had difficulties expressing themselves in English. They seem to lack the vocabulary necessary to describe their points of view.

However, one other point that should be reported is that during interviews, three students took the opportunity to criticize their teachers concerning their ways of teaching and their attitudes towards the students. They complained that the teachers did not help them enough in the laboratories because they were told that they were supposed to know the materials before coming to class. They were afraid to ask questions in class. They were sometimes left to themselves during those periods. Some students lost their motivation and did not concentrate much on the course.

All of this resulted in some students failing the introductory course or dropping it or abandoning the computer science program. It would be interesting to look at the various ways computer programming is being taught and how they can be accommodated to our Vanier College students. Thus a good combination of teaching and learning may assure success. Further research will tell.

BIBLIOGRAPHICAL REFERENCES

Bergin, S. & Reilly, R. (2005). *Programming: Factors that influence success.* SIGCSE'05, February 23-27, 2005 St. Louis, Missouri, USA.

Bergin, S. & Reilly, R. (2005). *The influence of motivation and comfort-level on learning to prgram.* Proceedings PPIG 17, June 2005 Sussex University, Sussex, UK, 2005.

Booth, S. (1992). *Learning to program: a phenomenographic perspective.* Acta Universitatis Gothoburgensis, Goteborg.

Bruce, C., McMahon, C., Buckingham, L., Hynd, J. & Roggenkamp, M. (2003). *Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university.* Retrieved: August 1, 2005. Available: http://eprints.qut.edu.au/archive/00001756/

Byrne, P., Lyons, G. (2001). *The Effect of Student Attributes on Success in Programming.* Paper presented at ITiCSE 2001, June 2001, Cantebury, UK, 2001.

Carbone, A & Kaasboll, J. J. (1998). *A survey of methods used to evaluate computer science teaching.* Paper presented at ITiCSE '98, Dublin, Ireland, 1998

Carrington, D. (1998). *Time monitoring for students.* Frontiers in Education Conference, IEEE, 1998.

Carter, J., Jenkins, T. (1999). *Gender and Programming: What's Going On?* Paper presented at ITiCSE '99, June 99, Cracow, Poland, 1999.

Du Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway & J. C. Spohrer (Ed.), *Studying the novice programmer* (pp.283-299). Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Freebody, P. (2003). *Qualitative research in Education, Interaction and Practice*, SAGE Publications, London.

Goold, A., Rimmer, R. (2000) *Factors Affecting Performance in First-year Computing*. SIGCSE Bulletin, vol. 32, No. 2, June 2000.

Hagan, D. & Markham, S. (2000). *Does it help to have some programming experience before beginning a computing degree program?* Proceedings of ITiCSE 2000, (pp 25-28).

Holmboe, C., McIver, L. & Carlisle, G. (2001). *Research Agenda for Computer Science Education*. Paper presented at the 13th Workshop of the Psychology of Programming Interest Group, Bournemouth, UK, April 2001, (pp 207-223).

Jacobs, J. (1887). Experiments on 'comprehension'. Mind, 12, 75-79.

Lahtinen, E., Ala-Mutka, K., Jarvinen H. (2005). *A Study of the Difficulties of Novice Programmers*. Paper presented at the ITiCSE'05, June 27-29, Monte de Caparica, Portugal, 2005.

Linn, M. C. & Dalbey, J. (1989). Cognitive consequences of programming instructions. In E. Soloway & J. C. Spohrer (Ed.), *Studying the novice programmer* (pp.57-81). Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Mancy, R., Reid, N. (2004). *Aspects of Cognitive Style and Programming*. Paper presented at the 16th Workshop of the Psychology of Programming Interest Group, 2004.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). *Report by the ITiCSE 2001 Working Group on Assessment of Programming Skills of First-year CS Students*. Retrieved:August8,2005.Available: http://coweb.cc.gatech.edu/guzdial/uploads/18/assessmentwg_iticse_final.pdf

Wilson, B. C. & Shrock, S. (2001). *Contributing to success in an introductory computer science course: a study of twelve factors*. Proceedings of the 23rd SIGCSE technical Symposium on Computer Science Education, ACM Press, NY, pp. 184-188.

Witkin, H. A., Moore, C. A., Goodenough, D. R. and Cox, P. W. (1977). Field dependent and independent cognitive styles and their educational implications. Review of Educational Research, 47, 1-64

APPENDIX A

THE SURVEY QUESTIONNAIRE

Name: ________________________________ **Student ID:** ___________

Questionnaire

The purpose of this questionnaire is to find out about your ways of learning to program and the factors involved in your learning process that will lead to the successful completion of your programming course. This survey may suggest ways to improve the teaching of computer science courses. That is why your answers are important to us. We assure you that **your responses will remain strictly confidential.**

Please follow the instructions very carefully. Thank you for your cooperation.

PART I

For each question or statement, circle the letter that provides the best answer.

1. What gender are you?
   a. female
   b. male

2 I have access to a computer at home.
   a. yes
   b. no

3. What is your current semester in the program?
   a. first
   b. second
   c. third
   d. other

4. Is English your
    a. first language
    b. second language

5. Did you do your high school studies in
    a. English
    b. French
    c. Other?

6. How I would rate my prior knowledge of how computers work before taking the course:
    a. very good knowledge
    b. good knowledge
    c. fair knowledge
    d. rather poor knowledge
    e. no knowledge

7. I have done some basic computer programming before.
    a. very much
    b. much
    c. a little
    d. very little
    e. not at all

8. I use the text books recommended by the teacher in order to understand the material better.
    a. all the time
    b. often
    c. sometimes
    d. rarely
    e. never

9. I analyze a problem and write the algorithm to solve it on paper before running the program.
    a. all the time
    b. often
    c. sometimes
    d. rarely
    e. never

10. I work in a group when I am doing my assignments or prepare for a test.
    a. all the time
    b. often
    c. sometimes
    d. rarely
    e. never

11. I work in a group when I prepare for a test.
    a. all the time
    b. often
    c. sometimes
    d. rarely
    e. never

12. I participate in class by asking questions.
    a. all the time
    b. often
    c. sometimes
    d. rarely
    e. never

13. I miss classes.

    a. never
    b. rarely
    c. sometimes
    d. often
    e. almost all the time

14. I have difficulties in understanding the concepts of programming.

    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

15. I have difficulties in problem solving.

    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

16. I find computer science difficult.

    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

PART II

Please, answer the following questions.

17. What was your average grade in final year of high school? _____

18. What was your high school final mark in mathematics 536 or equivalent?

    _____

19. If you have a job, how many hours a week do you work, on average?

    _____

20. What is the total number of hours per week do you spend studying on average, for all your college courses?

    _____

21. How many hours per week do you spend on average on studies related to your computer programming course?

    _____

22. What mark did you obtain on the first test (midterm) of the computer programming course?

    ____%

PART III

Questions about students' learning process

23. I try to understand the programming concepts before using them.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

24. I try examples in text books to help me understand the concepts when learning how to code.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

25. I learn the syntax of the programming language before applying it to solve a problem.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

26. I learn programming by analyzing and testing working programs.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree

d. strongly agree
e. don't know

27. I analyze problems and write the algorithms before attempting to do some coding.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

28. I usually do my assignments discussing with my peers in groups.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

29. I learn to program by copying or emulating others who are more experienced.
    a. strongly disagree
    b. somewhat disagree
    c. somewhat agree
    d. strongly agree
    e. don't know

30. I learn best by simply doing all the assigned assessment tasks.
    a. strongly disagree
    b. disagree
    c. agree
    d. strongly agree
    e. don't know

31. Which of the following describes you the best, when you learn programming? (Choose only one)

a. I learn the syntax of the programming language first and then spend a lot of time at the computer testing and running programs
b. I need to understand the concepts before I can apply them to practical tasks. I need to understand a concept fully before learning others
c. I start by analyzing a problem and then look at the concepts and syntax necessary to solve it
d. I gain experience and learn what programming is all about by learning the cultures and the ways of thinking of experienced programmers and try to follow their example
e. I learn by trying to do all the assessment tasks that are put of the course requirements.
f. I do not know.

PART IV (Open-ended question)

32. Can you elaborate on how you do a programming assignment? Is there a set of steps that you usually take to complete it? Please explain.

Thank you for filling out this questionnaire. Your responses are very helpful to all of us.

APPENDIX B

PROCEDURES FOR THE INTERVIEW

Procedures for the interviews

Students who were interviewed were chosen among the computer science students. They were chosen on the basis of gender and the answers from the survey. The main factor that was used for the choice was their way of learning. I ensured that each of the ways described by Booth and Bruce et al. was represented.

At the beginning of the interview, the students were explained the purpose of the study and how the results could be used to improve the teaching and learning of programming at school.

The students were told that all the data collected during the interview would be strictly used for the purpose of the study only. The personal data would be kept confidential and the results would be reported without bias and only to the appropriate parties.

The following questions would be used for the interviews.

- Tell me why you chose computer science.
- What is a programming language to you?
- What are the main techniques that you use when you learn to program?
- What type of assessment tasks help you most in learning to program?
- Can you describe how you go about writing a program?
- How do you overcome the frustration when your program is not working?
- Is there anything that you would like to see improved in how programming is being taught?

APPENDIX C

THE CONSENT FORM

CONSENT TO PARTICIPATE IN

"FACTORS INFLUENCING SUCCESS IN INTRODUCTORY PROGRAMMING COURSES"

I, the undersigned, agree to participate in the research project conducted by Pit F. Lan Chow Wing, a student in the Master Teacher Program given in collaboration with the Université de Sherbrooke.

I was informed that the purpose of the research project is to look at the factors that play a role in student success in introductory programming courses for computer science and science students. The goal of the research is to look at how students learn programming and what are the difficulties they encounter. Teachers can thus change their strategies to better meet the needs of the students.

I was informed that the data collected will remain confidential and that they will in no way affect my academic record at CEGEP.

I understand that I may be interviewed and that if so, the interview will be recorded.

I understand that the researcher can have access to my student records held by the Office of the Registrar.

I understand that even if I decide to participate at this time, I can subsequently change my mind and withdraw from the study. In such a circumstance, all the data I have contributed will be removed and my withdrawal will not affect my academic standing in any way.

I understand that the data collected for this study can be published but that my identity will remain confidential.

Date: ________________________

Print name (Given name, Family name):

____

Student no.: ___________________

e-mail address: _________________

Signature: _____________________

Signature (parent or guardian): _______________________

(If you are less than 18 years old)

APPENDIX D

FREQUENCY TABLES

Table 5
student gender

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	female	8	13.8	13.8	13.8
	male	50	86.2	86.2	100.0
	Total	58	100.0	100.0	

Table 6
accesscomputer

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	yes	58	100.0	100.0	100.0

Table 7
program of study

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	computer science	34	58.6	58.6	58.6
	science	24	41.4	41.4	100.0
	Total	58	100.0	100.0	

Table 8
mother tongue

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	first	25	43.1	43.1	43.1
	second	33	56.9	56.9	100.0
	Total	58	100.0	100.0	

Table 9
high school language

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	first	29	50.0	50.0	50.0
	second	23	39.7	39.7	89.7
	other	6	10.3	10.3	100.0
	Total	58	100.0	100.0	

Table 10
prior computer knowledge

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	no knowledge	3	5.2	5.2	5.2
	rather poor knowledge	6	10.3	10.3	15.5
	fair knowledge	16	27.6	27.6	43.1
	good knowledge	30	51.7	51.7	94.8
	very good	3	5.2	5.2	100.0
	Total	58	100.0	100.0	

Table 11
prior basic programming

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	not at all	28	48.3	48.3	48.3
	very little	10	17.2	17.2	65.5
	a little	18	31.0	31.0	96.6
	much	2	3.4	3.4	100.0
	Total	58	100.0	100.0	

Table 12
use recommended text

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	never	5	8.6	8.6	8.6
	rarely	7	12.1	12.1	20.7
	sometimes	17	29.3	29.3	50.0
	often	15	25.9	25.9	75.9
	all the time	14	24.1	24.1	100.0
	Total	58	100.0	100.0	

Table 13
analyze and write algorithm

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	never	7	12.1	12.1	12.1
	rarely	22	37.9	37.9	50.0
	sometimes	18	31.0	31.0	81.0
	often	9	15.5	15.5	96.6
	all the time	2	3.4	3.4	100.0
	Total	58	100.0	100.0	

Table 14
work in group doing assignment

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	never	8	13.8	13.8	13.8
	rarely	16	27.6	27.6	41.4
	sometimes	15	25.9	25.9	67.2
	often	19	32.8	32.8	100.0
	Total	58	100.0	100.0	

Table 15
work in group preparing test

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	never	25	43.1	43.1	43.1
	rarely	17	29.3	29.3	72.4
	sometimes	11	19.0	19.0	91.4
	often	5	8.6	8.6	100.0
	Total	58	100.0	100.0	

Table 16
participate in class

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	never	7	12.1	12.1	12.1
	rarely	17	29.3	29.3	41.4
	sometimes	24	41.4	41.4	82.8
	often	8	13.8	13.8	96.6
	all the time	2	3.4	3.4	100.0
	Total	58	100.0	100.0	

Table 17
miss classes

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	never	15	25.9	25.9	25.9
	rarely	28	48.3	48.3	74.1
	sometimes	13	22.4	22.4	96.6
	often	2	3.4	3.4	100.0
	Total	58	100.0	100.0	

Table 18

difficulties understand concepts

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	10	17.2	17.2	17.2
	somewhat disagree	27	46.6	46.6	63.8
	somewhat agree	17	29.3	29.3	93.1
	strongly agree	3	5.2	5.2	98.3
	don't know	1	1.7	1.7	100.0
	Total	58	100.0	100.0	

Table 19

difficulties solving problems

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	13	22.4	22.8	22.8
	somewhat disagree	30	51.7	52.6	75.4
	somewhat agree	10	17.2	17.5	93.0
	strongly agree	3	5.2	5.3	98.2
	don't know	1	1.7	1.8	100.0
	Total	57	98.3	100.0	
Missing	System	1	1.7		
Total		58	100.0		

Table 20

find computer science difficult

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	16	27.6	27.6	27.6
	somewhat disagree	24	41.4	41.4	69.0
	somewhat agree	13	22.4	22.4	91.4
	strongly agree	2	3.4	3.4	94.8
	don't know	3	5.2	5.2	100.0
	Total	58	100.0	100.0	

Table 21

understand programming concepts before using them

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	2	3.4	3.4	3.4
	somewhat disagree	1	1.7	1.7	5.2
	somewhat agree	27	46.6	46.6	51.7
	strongly agree	27	46.6	46.6	98.3
	don't know	1	1.7	1.7	100.0
	Total	58	100.0	100.0	

Table 22

try examples to understand the concepts of coding

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	9	15.5	15.5	15.5
	somewhat disagree	9	15.5	15.5	31.0
	somewhat agree	20	34.5	34.5	65.5
	strongly agree	20	34.5	34.5	100.0
	Total	58	100.0	100.0	

Table 23

learn syntax before appying it to solve problem

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	somewhat disagree	7	12.1	12.1	12.1
	somewhat agree	28	48.3	48.3	60.3
	strongly agree	21	36.2	36.2	96.6
	don't know	2	3.4	3.4	100.0
	Total	58	100.0	100.0	

Table 24

learn by analyzing and testing working programs

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	2	3.4	3.4	3.4
	somewhat disagree	13	22.4	22.4	25.9
	somewhat agree	13	22.4	22.4	48.3
	strongly agree	29	50.0	50.0	98.3
	don't know	1	1.7	1.7	100.0
	Total	58	100.0	100.0	

Table 25
analyze problem and write algorithm before coding

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	7	12.1	12.1	12.1
	somewhat disagree	20	34.5	34.5	46.6
	somewhat agree	27	46.6	46.6	93.1
	strongly agree	4	6.9	6.9	100.0
	Total	58	100.0	100.0	

Table 26
do assignments discussing with peers in groups

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	6	10.3	10.3	10.3
	somewhat disagree	13	22.4	22.4	32.8
	somewhat agree	21	36.2	36.2	69.0
	strongly agree	18	31.0	31.0	100.0
	Total	58	100.0	100.0	

Table 27
copy or emulate more experienced

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	strongly disagree	21	36.2	36.2	36.2
	somewhat disagree	13	22.4	22.4	58.6
	somewhat agree	19	32.8	32.8	91.4
	strongly agree	5	8.6	8.6	100.0
	Total	58	100.0	100.0	

Table 28
simply do all the assessment tasks

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	somewhat disagree	10	17.2	17.2	17.2
	somewhat agree	33	56.9	56.9	74.1
	strongly agree	14	24.1	24.1	98.3
	don't know	1	1.7	1.7	100.0
	Total	58	100.0	100.0	

Table 29
ways of learning

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	syntax and testing	14	24.1	24.6	24.6
	understanding concepts before applying	13	22.4	22.8	47.4
	analyze and look at concepts	16	27.6	28.1	75.4
	learn cultures of experienced programmers	3	5.2	5.3	80.7
	do all assessment tasks	6	10.3	10.5	91.2
	don't know	5	8.6	8.8	100.0
	Total	57	98.3	100.0	
Missing	System	1	1.7		
Total		58	100.0		

APPENDIX E

THE INTERVIEW: THE LEARNING PROCESS

Category ------->	Coding	Understang\	Problem Solving	Participation	Examples	Miscellaneous
	1	2	3	4	5	6
Analyze the problem and write algorithm			x			
Figure what I need to solve the problem and write the code			x			
Find the main concepts, main tools	x					
Try to understand the syntax and try examples in the book		x				
What is asked and try to create class			x			
Look at the syntax learned in class and try to find functions to complete the task		x				
Draw a picture of how the program works						x
Understand the problem and inspire from other programs					x	
Analyze the problem and write the code			x			
Carefully read the given problem and write the code			x			
Identify the variables and calculations required to solve it; try to figure out how to get into the code			x			
Think first about what I am going to need; then I start writing the program		x				
Try to see what kind of concept I am applying		x				
Look at the syntax after I know what I need to solve it	x					
Get confirmation from teacher or classmates				x		
Compare it with older assignments and use them as guidelines					x	
Read the book and some examples; clues and pointers given by teachers help		x				
Discuss with others; read the book and notes				x		
Read the chapter; read the assignment and write the methods		x				
Read all the chapters and refer to them for the assignment		x				
Read the corresponding chapters that relate to the program; do all the examples; try to understand the concepts		x				
Know all the theory needed for the assignment; write the code		x				
Read the instructions and highlight the important information; start coding			x			
Read the instructions and understand what is asked; start programming			x			
Read the problem; write all the variables; continue with the coding	x					
Read the instructions and evaluate the important information; type the program			x			
Understand the problem and look up at the syntax	x					
Divide the task into smaller tasks						x
Listen to the teacher on theory classes; try to do the program; compare my program with others				x		
Understand the problem; start coding			x			
Analyzing the problem; look at the concepts and syntax	x	x				

Look at the problem; identify the concept needed; ask a friend if I don't understand		x				
Set up a plan; write it						x
Write out the code; make sure concepts and syntax work by testing	x	x				
I look at what needs to be done; work on one problem at a time	x					
Try to understand the problem and write the commands I would need			x			
Look at the input needed, the output needed and the processing and code						x
Read what is given and what is required; write program			x			
Write the methods and test them; ask a friend or teacher if there are errors and got stuck	x					
Just read the book and try to understand what I am doing		x				
Use the textbook; try the examples in the book					x	
I put my head what needs to be done; write the steps	x					
Use examples of similar classes/methods; code the problem					x	
I think about what I need; write it						x
I read the assignment and break it into steps; consult more experienced programmer				x		
Read the program and think about the logic; code it		x				
I usually follow my class notes		x				
Try to understand the problem; ask my peers when I got stuck				x		
Basic understanding of the syntax; attempt the program; read textbook for more details	x					
Observe other examples; observed other programmers				x		
Look at the problem; what are the concepts that should be applied; write the code		x				

APPENDIX F

THE INTERVIEW: THE DIFFICULTIES ENCOUNTERED BY STUDENTS

Question 4	What are the difficulties encountered?	Concepts	Time factor	Problem solving	Relying on the book	Miscellaneous
S1						
1	First semester, I still did not understand the concepts that well. I was kind of scared. I did not really like it.	x				
S2						
S3						
S4/S5						
2	When the teacher teaches on the blackboard, you can't really see if the code is working.					x
3	We still do not understand some concepts.	x				
4	We cannot finish the long assignments before the due date.		x			
S6						
5	I can't solve the problems.			x		
S7						
6	I have lots of difficulties when I opened the book.					
7	You need to step away from the book, you need to step away from the computer.				x	
8	You have to figure out what is the problem and think of the solution.			x		
S8/S9						
9	I guess we did not know the logics and how to proceed with the problems.			x		
10	At the beginning I don't know what I am typing. I was just typing what I was told to type.					x
11	The problem that I have right now is to find what is the problem.			x		

S10						
12	It wasn't the greatest to tell you the truth. At the beginning of the semester I actually did pretty good but the problem was that you have to be consistent to keep up ...					x
13	I wasn't ... keeping practising it.					x
14	I was too dependent on the book.				x	
15	I fell behind because of the other courses.		x			
16	I had with that (that means problem solving).			x		

APPENDIX G

THE INTERVIEW: HOW THE STUDENTS COPE WITH THE DIFFICULTIES

Question 5	How do you cope with the difficulties encountered?				
S1					
1	I just keep trying over and over again.				
2	Sometimes I just stop and go and ask my teacher where I am doing wrong.				
3	Focus and I keep working.				
S2					
4	It's basically trial and error.				
5	I just find what is wrong with it looking what's wrong ... and then oe by one fix all the errors.				
6	Start with errors in compiling.				
7	Look at run time.				
S3					
8	To memorize some little words ...				
9	When you make a mistake in the program, you have to start from the beginning. But ...				
10	The method is basically a way to solve the problem. When you make a mistake, it's only in one method.				
S4/S5					
11	We get help from other students who actually know what they are doing.				
S6					
12	It's best to ask questions to the teacher or classmates.				
13	If you don't ask, you don't ... you can't solve the problem.				
S7					
14	Most of the time, ..., I take a break.				
15	I think over it again, go back to the first step I see if I made an error in the logic and see if I can break down the logic even more, ...				
16	And if that does not work, then I call a friend.				
S8/S9					
17	I just atake a break from it, do something else for a while. Then come back and it might become a bit more clear.				
18	I ask other students for help.				

S10					
19	I try to cram. The frustration is more with the other courses.				
20	It's not so hard if you put the time to it.				