Chapter 11
Enhancing the Metacognitive Skill of Novice Programmers Through Collaborative Learning

Margaret Bernard and Eshwar Bachu

Abstract Computer Supported Collaborative Learning (CSCL) aims to improve education by combining collaborative learning with modern information and communication technology. The opportunity exists to develop successful CSCL applications due to the increase in popularity of social networking and online gaming among students. In this chapter, we present an approach for promoting metacognition in computer programming using collaboration and computer games. We show that CSCL can improve the students' metacognitive skills and the use of games motivates and engages students in the learning process. Together, they enhance the qualities of a successful problem solver and low problem solving skill has been identified as the main challenge faced by novice computer programmers.

Keywords CSCL · Programming · Metacognition · Problem solving · COPS · Multiplayer game · Collaborative learning

Abbreviations

CL	Collaborative learning
COPS	Collaborative online problem solving
CSEC	Caribbean secondary education certificate
CSCL	Computer supported collaborative learning
CXC	Caribbean examinations council
DIV	Division

M. Bernard (✉) · E. Bachu
Department of Computing and Information Technology, The University of the West Indies, Circular Road, St. Augustine, Trinidad and Tobago
e-mail: margaret.bernard@sta.uwi.edu

E. Bachu
e-mail: eshwar.bachu@sta.uwi.edu

A. Peña-Ayala (ed.), *Metacognition: Fundaments, Applications, and Trends*,
Intelligent Systems Reference Library 76, DOI 10.1007/978-3-319-11062-2_11

ICT Information and communications technology
IT Information technology
MOD Modulo

11.1 Introduction

Teaching and learning of computer programming is a major challenge worldwide. A significant contributor to that challenge is the low problem solving ability of students. Students may understand the individual programming building blocks such as an 'if' statement or a 'while' loop but have difficulty in knowing how and when to use them.

Metacognition is a complex concept that relates to the higher order thinking that enables students to understand, analyze and control their own thought processes. This skill is particularly important in developing good programmers. Problem solving certainly involves cognition but more is required: students must constantly reflect on their strategies for problem solving and critically appraise their approach, thereby improving their connections between concepts. They need to build their knowledge of how and when to use particular strategies for problem solving.

Collaborative Learning has proved to be useful in many disciplines. In Computer Programming, the most commonly used strategy that involves collaboration is pair-programming; here students work in pairs, encouraging and correcting each other. A relatively new area of research is Computer Supported Collaborative Learning, in which the computer system is directly supporting the collaboration.

Bachu and Bernard [1] have shown that CSCL can increase the benefits of collaboration by enhancing the metacognitive abilities of students in the problem solving stage of programming. They present a framework for the development of a CSCL environment that incorporates a number of characteristics: the environment must promote positive interdependence where each member of the group becomes personally responsible for the group's success.

It should promote argumentative discussion where each member of the group must be aware of the need to make the best decision and encouraged to discuss and defend their reasoning for a given action, and it should promote equal participation where all members of the collaborative group take full responsibility for their learning and learn through the experiences of the other members of the group. Such systems aim directly at enhancing the metacognitive skill of students so that they know what they know and know how and when to apply basic concepts in a constructive manner to produce an algorithm that is the solution to a problem.

In this chapter, in Sect. 11.2 we first present a detailed description of the problem. We give an overview of some of the research work that has been done on analyzing and addressing the challenge of poor programming, particularly of novice

programmers. We put this section into a context that is the Caribbean region, where all countries have adopted a common exam. In Sect. 11.3 we discuss the issue of metacognition in programming.

We review the significant work of others in this area and develop and discuss three programming problems that illustrate the challenges students have with the problem solving phase of programming. Section 11.4 addresses an approach for promoting metacognition in programming using collaboration and computer games. Again we review the literature in this area of Collaborative Learning and Computer Supported Collaborative Learning, particularly as they relate to programming.

In Sect. 11.5, we present some important elements of a CSCL game environment that specifically targets enhancing metacognitive skills of the players. The game is a multi-player, turn-based game that encourages students to collaboratively build an algorithm that is the solution to a given problem. Section 11.6 presents the experimental results and findings from the use of the CSCL game. We conclude in Sect. 11.7 with some thoughts for future development and research.

11.2 Description of Problem

The teaching and learning of programming has posed a major challenge for educators worldwide for many decades; most students are able to learn basic programming skills but do not achieve any level of programming fluency [2]. In Trinidad and Tobago and the wider Caribbean region, students are introduced to computer programming at the secondary school level while preparing to sit the Caribbean Secondary Education Certificate (CSEC) Information Technology (IT) exam.

Prior to 2010, IT was offered in both General and Technical Proficiencies, but from 2010, all students were required to sit the General Proficiency exam. Table 11.1 presents the number of students who sat the General and Technical exams between 2005 and 2009.

The percentages in Table 11.1 indicate that only 4 % candidates sat the general exam prior to 2010. The main reason for the small number of general candidates was that the general exam placed greater emphasis on computer programming which the students had significant difficulties with. As a result, most schools allowed their students to sit the technical exam which contained very little

Table 11.1 CSEC IT candidate figures for June exam sittings [6]

Year	General	Percent (%)	Technical	Percent (%)	Total
2005	762	3.6	20,511	96.4	21,273
2006	898	3.8	22,446	96.2	23,344
2007	980	4.0	23,775	96.0	24,755
2008	1,210	4.4	26,064	95.6	27,274
2009	1,106	3.8	27,706	96.2	28,812

programming. From 2010, the lone general exam attempted to strike a balance in the programming requirements.

In 2010, CSEC IT had a pass rate of 84 and 79 % in 2011; while these are acceptable pass rates, a more in depth look at the examiners' reports produced for each exam sitting raises some major concerns in the areas of problem solving and programming in Pascal. The following are excerpts from these reports:

"Part (c) was poorly done as the majority of candidates could not write the correct algorithm for the given rule."; "Many candidates avoided this question. The use of arrays in programming challenged candidates" [3].

"The question was poorly done by the majority of candidates. Many candidates could not identify and correct the errors in the programming statements given in Part (a)"; "This question tested candidates' ability to use control structures and their knowledge of terms and concepts associated with programming. It was poorly done by the majority of candidates" [4].

"Part (c) was poorly done by the majority of candidates; they could not provide a correct arithmetic statement to calculate the final price"; "Part (a) was poorly done by the majority of candidates who appeared unfamiliar with the use of loops" [5].

"This question tested candidates' knowledge of concepts associated with problem solving and programming. Many candidates did not have a clear understanding of the concepts required to answer the various parts of the question" [7]. "Part (a) of the question was poorly done by the majority of candidates who were unable to identify the correct line numbers containing input, declaration and output statements. Part (b) was also poorly done. The majority of candidates could not identify the errors in the program segment and hence, could not provide the corrected codes. Candidates did badly on Part (c) as well. The majority of them seemed unfamiliar with the concepts of variable, data type and conditional statement" [8]. "The mean performance on this question was 3.75 out of a maximum of 15" [9].

These comments indicate that students experience significant difficulty with problem solving and programming. Beaubouef and Mason [10] also identified poor problem solving skill as a major contributing factor. There has been an abundance of research carried out to investigate the teaching and learning of programming; publications of this nature between 2005 and 2008 were analyzed by Sheard et al. [11] and suggestions which they offered as a way forward in programing education included the use of social networking and group work.

Problem solving requires reflecting on the solution, communicating the problem solution [12]; and the designing of a program to solve a particular task. Deek et al. [13] presented two challenges which students face when learning the task of program development: deficiencies in problem solving strategies and tactical knowledge; and ineffective pedagogy of programming instruction. They also presented a six step problem solving and program development model:

- Formulating the Problem: Preliminary Problem Description, Preliminary mental model, and Structured Problem Representation.
- Planning the Solution: Strategy discovery, goal decomposition, and data modelling.
- Designing the Solution: Organization and Refinement, Data/function specification, and Module Logic Specification.

- Translation: Implementation, Integration, and Diagnosis of Errors.
- Testing: Critical Analysis, Evaluation, and Revision.
- Delivery: Documentation, Presentation, and Dissemination.

The first three steps are those which present the toughest task for novices since it requires the problem solving ability which they lack. While problem solving remains their greatest challenge, there are other important skills which are necessary. Step four also requires novices to be able to comprehend and generate program code. Watson et al. [14] define programming comprehension as the ability to read and understand the outcomes of an existing piece of code and generation as the ability to create a piece of code that achieves certain outcomes. Achieving programming fluency would require developing students' problem solving and generation skills, however, this chapter focuses on increasing their problem solving ability. Recognizing these concerns, it becomes imperative to address the challenges which these students face. A repercussion of this is that students become disenchanted with computer programming at the secondary school level, and in the future, are hesitant to pursue to higher degrees in the fields Information Technology and Computer Science.

Their difficulties are also worrying since Sardone [15] recently highlighted that there is a need for producing college graduates who are considered to be fluent in information technology and programming lies at the core of information technology. Concerns about the high attrition rates in programming and computer science courses have also been raised [10, 16]. The authors of this chapter also have first-hand experience of the difficulties of introductory programming students.

11.3 Metacognition in Programming

Mayer [17] identified the importance of cognitive, metacognitive, and motivational skills in problem solving. Mayer argued that most students are able to solve routine problems (problems they are familiar with or have met before and know how to solve) but all three skills are vital for students to be able to solve non routine problems (problems they have failed to solve in the past or have never met). In the context of introductory programming, the basic cognitive skills can be broken down as: variables and constants; logical and comparison operators; selection statements; iterative statements; and arrays.

Mayer continued that mastering each of these component skill is not enough to promote non-routine problem solving, students need to know not only what to do, but also when to do it. He referred to this aspect of problem solving as metaskill or metacognition. Metacognition can take many forms; it includes knowledge about when and how to use particular strategies for learning or for problem solving [18].

Jonassen [19] describes the two main components of metacognition as knowledge of cognition and self-regulation. Knowledge of cognition requires knowledge of task requirements, knowledge of self (personal skills) and knowledge of learning styles. Self-regulation requires being able to monitor learning, plan

and select strategies and evaluation of regulation. Consider the next examples of programming problems:

1. Write a program which prompts the user to enter an integer and returns the sum of the digits in the number. E.g., if the user enter 123, your program should return 6 (1 + 2 + 3).
2. Write a program which tests each number between 20 and 60 to determine if it is even and prints the even numbers.

The first step in solving the above problems is to formulate the problem, i.e., to understand what the task requires and this presents a significant challenge to students [20]. Consider problem 1, while this may seem simple to understand and an example was given; students have experienced difficulty with understanding what is required especially when they have not met similar problems before.

Although the problem specifies that a single integer is to be inputted, a common misconception is that each digit will be inputted separately or that only three-digit integers will be inputted based on the example. Problem two requires that students be able to iterate through the numbers between 20 and 60 and determine the even numbers which are to be displayed, some students recognize that they know what the even numbers are and simply output/display them individually.

These fallacies are a result of the students' lack of determining task requirements which has been identified as key component of metacognition. Formulating or defining the problem would also entail correctly identifying the sub tasks involved in solving the problem in its entirety.

After correctly understanding what the problem requires, a solution must be derived and tested. In cases where the problem was misunderstood, it is possible to produce solutions which give correct results sometimes.

For problem two, a program which outputs all even numbers individually by adding 2 such as 20, 22, 24, …, 58, 60 would produce correct results each time the program is executed but it is still incorrect as it does not test the number to determine if it is even. Similarly a solution for problem one which expects only three-digit integers would be correct for those cases only. This highlights the importance of students being able to verify whether or not their solution is correct, however, developing this skill is difficult since the students' ability to evaluate their solution is linked to their knowledge of the task requirements. In practise, the verification of the solution is normally done by someone other than the developer.

In most cases, there are multiple correct solutions for a given problem and students are required to choose between them. At the introductory programming level, program efficiency is not mandatory but choosing the best solution should be encouraged. For problem one, the solution would entail performing a series of integer division (DIV) and modulo (MOD) of the inputted integer by ten (10). The modulo operation return the least significant digit and this would be added to a continuous sum while the integer division operation would remove the least significant digit from the integer until the integer is zero. The next pseudocode represents this solution:

```
Begin
  read n  /*let the inputted integer  be n*/
  sum = 0
  while n  <> 0
  sum = sum + (n MOD 10)
  n = n DIV 10
  end while
  print sum
End
```

Another possible solution would be to convert the inputted integer into a string (array of characters) and traversing through the array and summing all the digits. The following pseudocode represents this solution:

In the below pseudocode, TO_CHAR and TO_INT are pseudo-functions; students would be required to use the respective functions in the programming language they are using. There are also variations to the two solutions presented for problem one which are correct. Some students are able to recognize both solutions and would be required to choose between them; this relates to the component of metacognition regarding the planning and selecting of strategies.

```
  Begin
    read n /*let the inputted integer be n*/
  char[]nums = TO_CHAR(n) ; /* TO_CHAR converts the inte-
ger to array of characters*/
    i = 0
    sum = 0
    while i <  length(nums)
  sum= sum + TO_INT(nums[i]) /*TO_INT converts a charac-
ter to integer*/
      i = i + 1
    end while
    print sum
  End
```

Before the students can select a strategy, they must ensure that they possess the necessary programming skills to implement the solution this relates to the knowledge of self (personal skills) component of metacognition.

To implement the first solution, students need to know about the usage and purpose the DIV and MOD operations and about repetition loops (while, for, etc.). For the second solution, knowledge of repetition loops, arrays and the conversion functions are necessary.

For the second problem, the solution requires the traversal of all the numbers between 20 and 60 using a loop and determining whether each number is even using the MOD operation (MOD 2) and checking if the result is equal to zero.

If the result is 0 (no remainder when divided by two), the number is even, this can be done by checking to see if the result of MOD 2 was not equal to 1 but this requires a further understanding that MOD 2 would only return one of two values (0, 1).

For both problems, although they are different, the MOD operator is useful. The use of the while loop was different in both problems. For problem one, the loop condition was 'while *n* <> 0', i.e., while *n* was not equal to zero and the loop variable 'n' was modified within the loop using the DIV operator.

For problem two, the loop condition was i < length(nums), i.e., while *i* was less than the length of the array nums and the loop variable i was modified within the loop by increasing its value by 1.

A key problem solving skill that students should possess is to be able to recognize problems they have previously done and be able to transfer their knowledge from solving other problems which utilized the same skills. This is often referred to as 'far transfer' of learning [21]. Consider the following problem:

3. The year is 2013, in a small company, the CEO appointed new managers of accounts, finance and sales. Accounts managers are appointed for 2-year terms, finance managers for 3-year terms and sales managers for 4-year terms. Write a program which determines the next year in which there will be new appointees to all three positions.

For problem three, students are required to determine the next year after 2013 in which the difference between that year and 2013 is exactly divisible by 2, 3 and 4. A repetition structure and the use of MOD operator are useful for this problem which suggests that students who would've done problems one and two should be able to solve problem three.

However, for problem three, good problem solving students would recognize that they do not need to check every successive year since the longest term is 4 years; they only need to check every 8 years after 2013 so the loop variable can be incremented by 4. Also, when incrementing by 4, the variable will always be exactly divisibly by 4 and all numbers which are exactly divisible by 4 will be exactly divisible by 2, so the only necessary check is to determine if the variable is exactly divisible for 3 as given in the following solution:

```
Begin
  int yrDiff=4
  while yrDiff MOD 3 <> 0
    yrDiff = yrDiff + 4
  end while
  newYr = 2013 + yrDiff
  print newYr
End
```

The ability of students to transfer knowledge from previous problems they have done to new problems represents their ability to regulate and their learning and develop their learning strategies. For all three problems, students also need to know about variables, arithmetic calculations, program input/output and program sequencing. Most of these can be easily taught with the exception of sequencing. Sequencing represents the order in which the various program components form the solution and is the most important part of the solution. In both given solutions

for problem one, if the initialization of the sum variable was done within the while loop, incorrect sums would be calculated.

After determining whether they lack any of the necessary skills to implement the solution, it is the students' responsibility to learn/acquire these skills. For introductory programmers, this normally requires the intervention of the teacher since the students are normally unaware of their deficiencies at the introductory level, however, as they increase their problem solving skill, they will be able to better monitor and control their learning. Finally, there are certain operations where the order in which they are done can vary and a correct result is still obtained; for example, in the second solution for problem two, the order of the 'i' and 'sum' variables can be interchanged without affecting the main purpose of the program. We propose four dimensions of metacognitive skills that are required by programmers:

- Properly understand and correctly interpret the problem and what is required.
- Determine the steps required to solve the problem and know the correct sequence of the steps.
- Identify whether they possess the required skills for a particular solution, choose between different solutions and choose the best solution.
- Correctly verify whether their solution is correct.

The main challenge for educators and researchers lies in developing teaching methodologies which addresses the metacognitive and motivational aspects of problem solving. Collaborative learning has been identified as one such opportunity.

11.4 Promoting Metacognition in Programming Using Collaboration and Computer Games

11.4.1 Collaborative Learning

Ben-Ari [22] suggests that programming concepts need to be actively acquired by students and cannot be directly transferred from instructor to student. The application of constructivism to the teaching of programming is a possible solution [23–25]. Constructivist theory proposes that knowledge is actively constructed by students throughout the learning process and not absorbed from teachers or textbooks.

The theory also suggests that knowledge construction is recursive; therefore students will continuously build on what they already know and can also build on the experiences of other students and their teachers. Collaborative learning is an instruction method which utilizes constructivism.

Collaborative learning is an instruction method in which students work in groups towards a common academic goal [26]. Panitz [27] distinguished collaborative learning and cooperative learning as follows:

"Cooperative learning is defined by a set of processes which help people interact together in order to accomplish a specific goal or develop an end product which is usually content specific."

"Collaborative learning (CL) is a personal philosophy, not just a classroom technique. In all situations where people come together in groups, it suggests a way of dealing with people which respects and highlights individual group members' abilities and contributions."

However, the terms cooperation and collaboration have always been used interchangeably in research conducted in the areas of collaborative learning or cooperative learning since both are founded on constructivist learning theory. Sardone's comparison [15] of the traditional and constructivist learning environments showed that constructivist learning environments where active learning strategies are used, negate the influence of preferred learning styles. This suggests that collaborative learning approaches can meet the learning preferences of all students. Some of the major achievements of collaborative learning are:

- Motivation: Students are driven by a reward or goal structure and the only way they can attain their personal goal is if the entire group succeeds so they would encourage the other members of the group.
- Social Cohesion: Students may care about the other members of the group and would therefore help and encourage them.
- Development: Students will be exposed to the viewpoints and explanations of their group members and this will enhance their own cognitive processes.
- Cognitive Elaboration: Students will be required to explain their contributions and decisions and by having to provide to these explanations in a social context, it can help clarify and reinforce their own thought processes.

Alavi [28] as reported by Jara et al. [29] stated that collaborative learning methods tend to encourage the construction of knowledge, deeper understanding, and greater skill development since students are engaged in dynamic learning. Roger and Johnson [30] presented the following criteria for tasks which are deemed applicable to collaborative learning:

- The task is complex or conceptual.
- Problem solving is desired.
- Divergent thinking or creativity is desired.
- Mastery or retention is important.
- Quality of performance is expected
- Higher level reasoning strategies and critical thinking are needed.

Clearly all the listed criteria apply to computer programming. Acknowledging this, and the former stated benefits of collaborative learning (specifically, motivation and cognitive elaboration), collaborative learning appears to be a tool which can be successfully utilized for teaching programming to novices.

Additionally, Kelleher and Pausch [31] identified a lack of social context in programming and suggested that the use of group work can make the task of learning programming easier and fun. Collaborative learning is also a more realistic model of how software is developed in industry as opposed to the solitary programming which is normally used in introductory programming courses [32].

Furthermore, most software development companies utilize software development methodologies like extreme programming which incorporate team work as a fundamental component. In extreme programming, managers, customers and developers are all equal partners in a collaborative team.

11.4.2 Collaboration Enhances Programming Metacognition

Encouraging novice programmers to collaborate can help alleviate some of the challenges which they face while programming. Through structured collaboration, the students will be forced to engage in argumentative discussion where they are required to listen to the viewpoints and opinions of their peers and offer their own.

Referring to the programming process, when the students collaborate, they can avoid misinterpreting of the task requirements since they can correct each other's misconceptions. The collaboration also increases the chances of multiple solutions being developed for the same problem and through argumentative discussion, the best one can be chosen.

The implementation of the solution also becomes easier since it is more likely that the all skills required to implement the solution exists within the group than for an individual programmer. Finally, the evaluation of the solution also becomes more effective since it can be tested by different persons within the group.

A study which investigated the usefulness of collaboration for Java programming concluded that collaboration was deemed most important while conceptualizing, brainstorming, and formulating the problem and its requirements; also the more complex the problem, the greater the importance of the collaboration [33].

This result follows with research which suggested that the major cause of students' failure in introductory programming is the lack of basic problem solving skill. Most of the research on the use of collaboration to teach programming is in the pair programming pedagogy. Pair programming is described as:

> A style of programming in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test. One of the pair, called the driver, types at the computer or writes down a design. The other partner, called the navigator, has many jobs. One is to observe the work of the driver, looking for defects. The navigator also has a more objective point of view and is the strategic, long-range thinker. Together, the driver and the navigator continuously brainstorm a solution. Periodically, the programmers switch roles between the driver and the navigator [34].

Davidson [35] identified the key attributes of collaborative learning as:

- Common Task or Learning Activity.
- Small Group Learning.
- Cooperative Behavior.
- Positive Interdependence.
- Individual Accountability and Responsibility.

Preston [36] used the above framework to analyze the pair programming pedagogy and concluded that pair programming is a model for collaborative learning. DeClue [37] concluded that pair programming has positive motivational characteristics; produces higher quality programs with coherent design and code documentation; leads to decreased time to complete assignments; and improves understanding of the programming and software engineering processes.

Collaborative learning was mainly adopted in classroom based environments which required face to face interaction between learners, as is the case with pair programming. This approach has shown to be very useful to learners, but it needs to be extended and enhanced to make its benefits more accessible to teachers and students.

11.4.3 Computer Supported Collaborative Learning (CSCL)

Computer Supported Collaborative Learning (CSCL) has been identified as one of the most promising innovations to improve teaching and learning with the help of modern information and communication technology (ICT) [38]. CSCL aims to enhance learning by combining computer support and collaborative learning [39].

Originally, collaborative learning was mainly adopted and confined to the classroom based environments which required face to face interaction between students and lecturer; by utilizing technology, there is no longer the need for this physical interaction. Through CSCL, the opportunity also exists to extend the learning process from the classroom and make it easily available to students.

Newman et al. [40] claims that a clear link between critical thinking, social interaction, and deep learning has emerged. An abundance of social interaction takes place on the Internet and therefore it is possible for good CSCL systems to do just as well in promoting learning as conventional group work.

A study conducted by Pifarre and Cobos [41] found an increase in students' metacognitive skills after using a CSCL system and their result was similar to other findings which the authors reported as part of their literature review. This suggests that the use of CSCL systems enhance the development of metacognitive learning processes. Lee et al. [42] concluded that while engaged in group or community learning, students analyzed the community discourse and improved their own understanding.

Diziol et al. [43] stated that when students collaborated on conceptual-problem solving steps, they talked to each other and provided mutual explanations. This led to improved learning when compared to individual learning. Chen [44] also proposed that the use of collaborative tools such as discussion boards or emails can be exploited to stimulate student motivation and encourage problem solving. These findings all indicate successful applications of CSCL for problem solving.

Argumentation and how students can benefit from it has always been a main focus in the field of CSCL [45]. Lu et al. [46] suggested that argumentation tools and visualization can be designed in CSCL to facilitate students' problem solving

by promoting collaboration and shared understanding. Collaboration Scripts have been identified as a solution for improving the quality of argumentation. A collaboration script is defined as:

"A set of instructions regarding to how the group members should interact, how they should collaborate and how they should solve the problem" [47]. Implementing CSCL scripts has resulted in improved learning outcomes [48, 49]. Scripts can also be useful in helping students structure their argumentative discourse [50].

11.5 Multiplayer Games to Support Programmers Problem Solving

11.5.1 Educational Multiplayer Games

Games have been known to create interest, cooperation, and competition for its players. These are all qualities which educators have strived to inspire in the classroom [51], therefore it makes sense to merge the motivation of games with learning.

Games provide a challenge and deliver rewards which encourage students to work harder and can be used to encourage learning. The long term effects of game playing are as follows [52]:

- Heightened concentration.
- Increased Intelligence.
- Batter hand-eye coordination.
- Increased stamina and determination.
- Better multi-tasking.
- Better awareness.

Doherty and Kumar [53] recognized the highly abstract nature of core programing concepts and suggested that games which are successful at teaching programming are those which cause the learner to develop and understand concepts from the content of the game as a consequence of its system and interface. Doherty and Kumar defined a game environment as one in which the concepts that emerge from interacting with it are created by the goal. Games can also help to alleviate some of the difficulties which students face while programming [54].

Recognizing the important benefits of collaboration in learning, researchers began exploring the possibility of building educational or collaborative multiplayer games. The possibility of using multiplayer games as educational tools has been explored using factors such as frequency of game play, gender, self-esteem, computer self-efficacy, and academic performance [55].

Their findings strengthened the possibility of multiplayer games becoming educational tools that can engage students and lead to accomplished learning outcomes. Li et al. [56] agreed that online multiplayer games can be used as educational tools if they are guided by an appropriate learning theory like constructivist theory.

It has been suggested that the principles of CSCL and problem solving can be applied to multiplayer games [57]. They put forward that the game mechanics should not only encourage but rather require players to engage in collaborative interactions in order to solve the problem; and that collaborative interactions should be enforced, rather than competitive ones.

11.5.2 Guidelines for Designing Multiplayer Games to Support Problem Solving

The following are guidelines for creating multiplayer games which aim to enhance the metacognitive skills of novice programmers. The game's design and characteristics should stimulate collaboration not only as part of the task, but as an integral part of the learning process. It should also promote collaborative rather than competitive interactions. Strategy games are preferred since it requires careful thinking and planning by the players to ensure success.

The two common types of strategy games are real-time strategy (RTS) and turn-based games. In turn-based games, each player is required to pay attention to the moves made by every other player, whereas RTS games allow players to play independently for portions of the game. In both cases, the game environment changes to reflect the actions performed by the players.

The game should engender or embody the educational content. A game which embodies educational content is one which has the educational content as a core part of its system; and a game which engenders educational contents causes the learner to develop and understand concepts from the content as a consequence of its system and interface [53]. The educational content should not be a simple addition to the game in the form of multiple choice or fill in the blank questions; instead, the game should be designed around the concepts being taught.

A game can motivate students to learn by rewarding learning, practice or mastery with in-game success [53]. This means that players cannot be successful at the game unless they understand the educational concepts which are being taught. A main challenge of many games is to earn rewards or get the highest score; students should not be allowed to attain these unless they are successful at the learning tasks in the game.

A game which requires the players to earn points to progress from one level to the next; or to unlock new features and get bragging rights can also be very useful in motivating students. The use of games can provide the motivation which is important to ensure that learning occurs and the students acquire the necessary skills.

However, it is important that the gameplay and aesthetics do not overwhelm the educational content. The learning tasks should always be the priority, the players should not be allowed to stray away from the required learning tasks. The game should implement a cooperative rather than a competitive reward structure. Players will work in groups to accomplish a learning task and all members of the group should be rewarded or penalized equally.

This ensures that the members of the group understand that their success lies in the success of the entire group and they can only succeed if the group as a whole succeeds; this is referred to as positive interdependence.

Positive interdependence also encourages the development and cognitive elaboration benefits of collaborative learning [58]. Members of the group would be more likely to help their group members and receive help in return; during this exchange they would be exposed to the viewpoints of other members and presented with the opportunity to express their thoughts, which in turn contributes to their development.

Promoting positive independence will also foster higher level critical thinking and reasoning strategies; and encourage a willingness to take on more difficult tasks [59]. Each member of the group becomes personally responsible for the team's success and would be encouraged to try harder because they know that their group members are dependent on them.

Argumentative Discussion is a key feature of the game design and it is related to positive interdependence. Each member of groups feels responsible for the team's success so they would make every effort to ensure they succeed; however, the game should ensure the group members are always aware of the need to make the best decisions throughout the entire game.

This can be enforced using time or other constraints. For example, the group can be required to accomplish an objective within a fixed time period, within a fixed number of moves, or to achieve an outcome which satisfies specified criteria.

Group members will be encouraged to compare and contrast their reasoning and decisions with their group members promoting higher quality decision making, creativity and problem solving.

This will also lead to greater productivity by the entire group since the members would strive to make the best possible decisions throughout the game.

Equal participation within a group is a vital area of concern since experiences with traditional group work have always shown cases where certain members of the group take full responsibility and the other members do not participate. The easiest approach for ensuring equal participation would be implementing a turn based approach. However, if complex tasks are being targeted, they can be divided into smaller tasks and each member can be assigned as the lead for a subtask.

Equal participation entails not only ensuring that each member works on their own task but that each member of the group has the opportunity to contribute to what the other members have done. Each activity or move made in the game should be a result of group collaboration.

In CSCL research, individual accountability refers to an individual evaluation after the collaborative process is completed. It suggests that in order to accurately evaluate the collaborative learning process, each group member must be individually assessed as they are responsible for their own learning.

However, in this context individual accountability is interpreted as encouraging each group member to explain their actions or moves to their group members to promote argumentative discussion. The game should create scenarios which

require each member to explain their decisions to their group members. In return, the members must assess and contribute to what was done. This helps to ensure that the best possible group decisions are made throughout the game.

11.6 Implementation and Experimental Findings

11.6.1 Implementation

The main output of problem solving is an algorithm. An algorithm is the sequence of steps required to solve a problem. Flowcharts and pseudocode are two common program design tools used for the representation of algorithmic solutions. Pseudocode is a text based representation which consists of English-like statements.

It is designed to fill the gap between the informal (spoken or written) description of the programming task and the final program code [60]. Flowcharts are a visual representation of program flow using a combination of arrows and symbols to represent the actions and sequence of the program.

Collaborative Online Problem Solving (COPS) is a turn based strategy game in which groups of two, three or four players are required to collaboratively build a program flowchart for a given problem within a target number of moves.

COPS was developed using the guidelines presented in the previous section. There are two different games in COPS:

- SWAP: The group is shown a flowchart with pieces out of order and the players are required to swap pieces to correct the flowchart. This game is designed to be the easier level of COPS.
- JIGSAW: The group is required to build a flowchart similar to how they would construct a jigsaw puzzle. This game is designed to be the harder level of COPS.

Each member of group receives 10 points for each SWAP game and 20 points for each JIGSAW game which is completed within the target number of moves. For both games, the group is deducted one point for each extra move they make beyond the target number of moves. For games which the group quits, no points are awarded.

While each member of the group is awarded equally, a player is allowed to play with different groups and increase the number of points which they earn. This allows players who play regularly to score more points and stimulates a competitive atmosphere amongst players.

Each problem in COPS has an associated question type. The question types are categorized based on the basic skills which novice programmers should acquire. Each puzzle also has an associated difficulty level. In order to progress to more difficult puzzles, players are required to reach a minimum number of points.

The collaboration in COPS is enforced through a voting system. COPS is turn based and each time a player makes a move, the other group members are required to vote on whether they agree or disagree with the move; if the move receives a majority vote, the game accepts the move otherwise it is rejected. To avoid ties, the player making the move is given a higher weighted vote.

The voting system aims to encourage argumentative discussion, positive interdependence and individual accountability. The player who makes the move would be required to explain their move and decisions using the chat system to their group members to convince them to accept the move. Similarly, the other group members would be required to explain why they may not agree with the moves.

The target number of moves is also meant to encourage the group make the best possible move each time since they only earn maximum points if they solve the puzzle within the target number of moves. The enforced collaboration between the group members ensures that every member of the group is involved in each game move and makes them accountable for their decisions. The overall design of COPS also ensures that the individual players only succeed when the entire group succeeds.

COPS provides intelligent feedback to players through graphical and textual alterations to help the students visualize their problem solving. For each accepted move, COPS automatically generates the pseudocode equivalent of the flowchart regardless of whether the flowchart is correct or incorrect.

The pseudocode guide is useful since it can help the players when they become stuck and it also shows them the pseudocode for their flowchart which will be useful in implementing the solution to the problem. Within the SWAP game, the places in the flowchart which are in incorrect positions are highlighted to guide the learning process; un-highlighted pieces indicate to the players that the pieces are correct and they can focus on solving other parts of the puzzle. In the JIGSAW game, the parts of the flowchart which are correct are highlighted to offer the same guidance to players.

Referring back to the definition of metacognition in programming given in Section three, COPS aims to help students in many ways. All problems in COPS are done by a group and through the chat system provided; members can help each other clarify any misinterpretations with the problem description. In both the SWAP and JIGSAW games, the group is given the general components which form the solution but it is the responsibility of the group to determine the sequence of the pieces/components to build the solution. COPS also uses intelligent feedback to let the players know whether they solution is correct or incorrect and accepts multiple correct solutions for the same problem, so the players are allowed to consider varying solutions.

11.6.2 Experimental Findings

The primary target users of COPS are secondary school students between the ages of 13 and 17 who are learning problem solving and programming for the first time. However, COPS focuses on problem solving and is language independent so it can be used for introductory programming courses at any level or institution. A survey of secondary school students was done asking about their difficulties with problem solving. The responses were categorised based on the four dimensions of programming metacognition given at the end of section three. The findings indicated that:

- 52 % of respondents admitted difficulty in understanding and interpreting a problem and what is required.
- 19 % had difficulty with determining the steps required to solve the problem and knowing the correct sequence of the steps.
- 7 % had challenges choosing between different solutions and choosing the best solution.
- 12 % admitted having problems with the syntax of the programming language.
- 10 % responded that they don't know what their major difficulties were.

The first statistic reiterates the findings of previous research and supports the case for the use of collaboration for teaching programming. Two studies were done to investigate the usefulness of COPS for secondary school students across Trinidad and Tobago in learning problem solving and programming.

A control version (non-collaborative/single player) of COPS was built as compared against the collaborative (multiplayer) version of COPS while being used by introductory programming students who had never done programming before.

An ancova analysis of the pre and post test results from study one showed significantly (p-value: 0.002) better performance by the students who used the collaborative version (mean improvement 14.27 %) than those who used the controlled version (mean improvement of 11.58 %). For the second study, there was no control, but the collaborative version of COPS was used by students who had previously done programming A paired t-test of the pre and post test results showed a significant (p-value: 0.000) improvement by the participants by 21.56 %.

The results from both studies indicated that COPS was useful both as a learning and revision tool for novice programmers. A more detailed analysis of the collaboration amongst participant in both studies showed that the first time programmers from the first study benefitted more from collaborating with the same students more often. However, the participants from the second study who had done programming before benefitted more from collaborating with different students. These findings indicate that COPS can be used to successfully enhance the problem solving skill of novice programmers.

11.7 Conclusion

The chapter addresses the challenge of problem solving in computer programming and presents an approach for Collaborative Learning that enhances the metacognitive skill of novice programmers. Many students acquire basic coding skills but they are unable to utilize them in a meaningful way to solve non-routine problems and they are unable to verify whether their solution is correct. Improving their metacognitive ability would help students identify and understand what a problem requires and analyse and evaluate the different alternative solutions to the problem.

Collaborative learning, which is founded on constructivist learning theory, has been shown to help students improve their problem solving skill by promoting

metacognition. In experiments conducted using a collaborative strategy game, students improved their metacognition skill.

They were better able to understand the requirements of the problem and, through visualizing the solution using flowcharts and pseudocode, improve in decomposing the problem into manageable chunks, all the while enjoying the interaction with other players. Students were captivated with the game and learning took place transparently. The chapter provides a comprehensive review of the state-of-the-art of research on Collaborative Learning and Metacognition.

We developed a framework for successful computer supported collaborative learning environments; the collaboration should encourage equal participation, argumentative discussion and positive interdependence.

We have developed a multiplayer strategy game which conforms to this framework and which improves the metacognition of each player. By enforcing equal participation, each player is motivated to understand the problem and analyse the logic in the programming solution; the argumentative discussion means that they must be able to defend their solution to the other players in the team and positive interdependence means that all players must learn if the team is to complete the game successfully and no player is left behind.

There are still several open areas for research and future work. One of these has to do with integration of CSCL environments into traditional classroom teaching on a large scale. Developing a truly blended approach is not a straightforward task. We are convinced that teachers and educators play an invaluable role in the success of CSCL games. The success of the games relies on its adoption and addition to the classroom. Secondly, future work could focus on the whole program development life cycle.

We have focused on the early phases of understanding the requirements of the problem and developing a solution. After solving a puzzle in COPS, the players/students would have an algorithmic solution (flowchart/pseudocode) for the problem but they are still required to write the program code afterwards. This has its own set of challenges. We also want to examine the relationships between different types of players collaborating in the group.

Do students prefer to play with other inexperienced players that they may know well or is there some benefit in playing with more advanced programmers? People are naturally drawn to people that they are comfortable with. We want to study how groups are formed online, the dynamics of the group, and the impact of different combinations of players with differing abilities.

References

1. Bachu, E., Bernard, M.: A computer supported collaborative learning (CSCL) model for educational multiplayer games. In: 11th International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government. Las Vegas (2012)
2. Bachu, E., Bernard, M.: Enhancing computer programming fluency through game playing. Int. J. Comput. **1**(3) (2011)

3. CXC. Information technology general proficiency examination May/June 2011. Report on Candidates' Work in the Secondary Education Certificate Examination. St. Michael, Barbados (2011)
4. CXC. Information technology general proficiency examination May/June 2010. Report on Candidates' Work in the Secondary Education Certificate Examination. St. Michael, Barbados (2010)
5. CXC. Information technology general proficiency examination May/June 2012. Report on Candidates' Work in the Secondary Education Certificate Examination. St. Michael, Barbados (2012)
6. CXC. Annual reports for year 2005–2009. St. Michael, Barbados
7. CXC. Information technology general proficiency examination January 2011. Report on Candidates' Work in the Secondary Education Certificate Examination. St. Michael, Barbados (2011)
8. CXC. Information technology general proficiency examination January 2010. Report on Candidates' Work in the Secondary Education Certificate Examination. St. Michael, Barbados (2010)
9. CXC. Information technology general proficiency examination January 2012. Report on Candidates' Work in the Secondary Education Certificate Examination. St. Michael, Barbados (2012)
10. Beaubouef, T., Mason, J.: Why the high attrition rate for computer science students: some thoughts and observations. ACM SIGCSE Bull. **37**(2), 103–106 (2005)
11. Sheard, J., Simon, S., Hamilton, M., Jan L.: Analysis of research into the teaching and learning of programming. In: 5th International Computing Education Research Workshop, pp. 93–104. ACM, New York (2009)
12. Gomes, A., Mendes, A.J.: Learning to program-difficulties and solutions. In: International Conference on Engineering Education, vol. 2007. Coimbra, Portugal (2007)
13. Deek, F.P., McHugh, J.A., Turoff, M.: Problem solving and cognitive foundations for program development: an integrated model. In: Sixth International Conference on Computer Based Learning in Science (CBLIS), pp. 266–271. Nicosia, Cyprus (2003)
14. Watson, R., de Raadt, M., Toleman, M.: Teaching and assessing programming strategies explicitly. In: 11th Australasian Computing Education Conference (ACE 2009), Wellington, New Zealand (2009)
15. Sardone, N.B.: Developing information technology fluency in college students: an investigation of learner environments and learner characteristics. Inf. Technol. Educ. **10**(1), 101–122 (2011)
16. Hundhausen, C.D., Farley, S.F., Brown, J.L.: Can direct manipulation lower the barriers to computer programming and promote transfer of training?: an experimental study. ACM Trans. Comput. Hum. Interact. **16**(3), 1–40 (2009)
17. Mayer, R.E.: Cognitive, metacognitive, and motivational aspects of problem solving. Instr. Sci. **26**(1), 49–63 (1998)
18. Metcalfe, J., Shimamura, A.: Metacognition: Knowing About Knowing. Bradford Books, Cambridge (1994)
19. Jonassen, D.: Learning to solve problems: A Handbook for Designing Problem-Solving Learning Environments. Taylor & Francis, United Kingdom (2011)
20. Bachu, E.: A Framework for Computer Supported Collaborative Learning (CSCL) Using Online Multiplayer Games. M.Phil., dissertation, The University of the West Indies, St. Augustine, Trinidad and Tobago (2013)
21. Viviene, C., Macaulay, C.: Transfer of Learning in Professional and Vocational Education. Psychology Press, United Kingdom (2000)
22. Ben-Ari, M.: Constructivism in computer science education. In: 29th SIGCSE Technical Symposium on Computer Science Education, pp. 257–261. ACM, New York (1998)
23. Gonzalez, G.: Constructivism in an introduction to programming course. J. Comput. Sci. Coll. **19**(4), 299–305 (2004)

24. Boyer, N.R., Langevin, S., Gaspar, A.: Self direction and constructivism in programming education. In: 9th ACM SIGITE Conference on Information Technology Education, pp. 89–94. ACM, New York (2008)
25. Lui, A.K., Kwan, R., Poon, M., Cheung, Y.H.Y.: Saving weak programming students: applying constructivism in a first programming course. ACM SIGCSE Bull. **36**(2), 72–76 (2004)
26. Gokhale, A.: Collaborative learning enhances critical thinking. J. Technol. Educ. **7**(1), 56–65 (1995)
27. Panitz, T.: Collaborative versus cooperative learning: a comparison of the two concepts which will help us understand the underlying nature of interactive learning (1999)
28. Alavi, M.: Computer-mediated collaborative learning: an empirical evaluation. MIS Q. **18**(2), 159–174 (1994)
29. Jara, C.A., Candelas, F.A., Torres, F., Dormido, S., Esquembre, F., Reinoso, O.: Real-time collaboration of virtual laboratories through the Internet. Comput. Educ. **52**(1), 126–140 (2009)
30. Roger, T., Johnson, D.W.: An overview of cooperative learning. In: Thousand, J., Villa, A., Nervin, A. (eds.) Creativity and Collaborative Learning. Brookes Press, Baltimore (1994)
31. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. **37**(2), 83–137 (2005)
32. Urness, T.: Assessment using peer evaluations, random pair assignment, and collaborative programing in CS1. J. Comput. Small Coll. **25**(1), 87–93 (2009)
33. Bagley, C.A., Chou, C.C.: Collaboration and the importance for novices in learning Java computer programming. In: 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, pp. 211–215. ACM, New York (2007)
34. Williams, L.: Lessons learned from seven years of pair programming at North Carolina State University. SIGCSE Bull. **39**(4), 79–83 (2007)
35. Davidson, N.: Cooperative and collaborative learning: an integrative perspective. In: Thousand, J., Villa, R., Nevin, A. (eds.) Creativity and Collaborative Learning: A Practical Guide to Empowering Students and Teachers, pp. 13–30. Paul H. Brookes Publishing Co., Baltimore, MD (1994)
36. Preston, D.: Pair programming as a model of collaborative learning: a review of the research. J. Comput. Sci. Coll. **20**(4), 39–45 (2005)
37. DeClue, T.H.: Pair programming and pair trading: effects on learning and motivation in a CS2 course. J. Comput. Sci. Coll. **18**(5), 49–56 (2003)
38. Ehtinen, E., Hakkarainen, K., Lipponen, L., Rahikainen, M., Muukkonen, H.: Computer supported collaborative learning: a review. The JHGI Giesbers Reports on Education (1999)
39. Stahl, G., Koschmann, T., Suthers, D.: CSCL: an historical perspective. In: Sawyer, K.R. (ed.) Cambridge Handbook of the Learning Sciences, vol. 5, pp. 409–426. Cambridge University Press, UK (2006)
40. Newman, D.R., Webb, B., Cochrane, C.: A content analysis method to measure critical thinking in face-to-face and computer supported group learning. Interpersonal Comput. Technol. **3**(2), 56–77 (1995)
41. Pifarre, M., Cobos, R.: Promoting metacognitive skills through peer scaffolding in a CSCL environment. Int. J. Comput. Support. Collaborative Learn. **5**(2), 237–253 (2010)
42. Lee, E.Y.C., Chan, C.K.K., Van-Aalst, J.: Students assessing their own collaborative knowledge building. Int. J. Comput. Support. Collaborative Learn. **1**(1), 57–87 (2006)
43. Diziol, D., Rummel, N., Spada, H., McLaren, B.M.: Promoting learning in mathematics: script support for collaborative problem solving with the cognitive tutor algebra. In: 8th International Conference on Computer Supported Collaborative Learning, pp. 39–41. ISLS, USA (2007)
44. Chen, J.W.: Designing a web-based Van Hiele model for teaching and learning computer programming to promote collaborative learning. In: 5th IEEE International Conference on Advanced Learning Technologies, pp. 313–317. IEEE, NJ (2005)

45. Stegmann, K., Weinberger, A., Fischer, F.: Facilitating argumentative knowledge construction with computer-supported collaboration scripts. Int. J. Comput. Support. Collaborative Learn. **2**(4), 421–447 (2007)
46. Lu, J., Lajorie, S.P., Wiseman, J.: Scaffolding problem-based learning with CSCL tools. Int. J. Comput. Support. Collaborative Learn. **5**(3), 283–298 (2010)
47. O'Donnell, A.M., Dansereau, D.F.: Scripted cooperation in student dyads: a method for analyzing and enhancing academic learning and performance. In: Hertz-Lazarowitz, R., Miller, N. (eds.) Interaction in Cooperative Groups: The Theoretical Anatomy of Group Learning, pp. 120–141. Cambridge Universirty Press, UK (1992)
48. Rummel, N., Spada, H.: Can people learn computer-mediated collaboration by following a script? In: Fischer, Frank, Kollar, Ingo, Mandl, Heinz, Haake, JörgM (eds.) Scripting Computer-Supported Collaborative Learning, pp. 39–55. Springer, USA (2007)
49. Weinberger, A., Fischer, F., Mandl, H.: Fostering computer supported collaborative learning with cooperation scripts and scaffolds. In: 5th International Conference on Computer Supported Collaborative Learning, pp. 573–574. ISLS, USA (2002)
50. Bures, E.M., Abrami, P.C., Schmid, R.F.: Exploring whether students' use of labelling depends upon the type of activity. Int. J. Comput. Support. Collaborative Learn. **5**(1), 103–116 (2010)
51. Prensky, M.: Digital game-based learning. Comput. Entertainment **1**(1), 1–4 (2003)
52. Tsiatsos, T.A., Konstantinidis, A.: Utilizing multiplayer video game design principles to enhance the educational experience in 3D virtual computer supported collaborative learning environments. In: 12th IEEE International Conference on Advanced Learning Technologies, pp. 621–623, IEEE, NJ (2012)
53. Doherty, L., Kumar, V.: Teaching programming through games. In: International Workshop on Technology for Education, pp. 111–113. IEEE, Bangalore (2009)
54. Rajaravivarma, R.A.: Games-based approach for teaching the introductory programming course. SIGCSE Bull. **37**(4), 98–102 (2005)
55. Paraskeva, F., Mysirlaki, S., Papagianni, A.: Multiplayer online games as educational tools: Facing new challenges in learning. Comput. Educ. **54**(2), 498–505 (2010)
56. Li, Y., Tian, X., Gao, P.: Research on the application of MMO games in education. In: International Conference on Industrial Control and Electronics Engineering (ICICEE), pp. 535–538. IEEE, NJ (2012)
57. Voulgari, I., Komis, V.: Massively multi-user online games: the emergence of effective collaborative activities for learning. In: 2nd IEEE International Conference on Digital Game and Intelligent Toys Based Education (DIGITEL), pp. 132–134. IEEE, Banff, BC (2008)
58. Slavin, R.E.: Research on cooperative learning and achievement: what we know, what we need to know. Contemp. Educ. Psychol. **21**(1), 43–69 (1996)
59. Johnson, D.W., Johnson, R.T., Karl Smith, K.: The state of cooperative learning in postsecondary and professional settings. Educ. Psychol. Rev. **19**(1), 15–29 (2007)
60. Roy, G.G.: Designing and explaining programs with a literate pseudocode. ACM J. Educ. Resour. Comput. **6**(1), 1–18 (2006)