The role of metacognitive skills in solving object-oriented programming problems: a case study

M HAVENGA[1]

Abstract

This article reports on the role of metacognitive skills when solving object-oriented programming problems as part of a case study. The research was constructivist-based within an interpretivist approach to explore how four students constructed their own thinking when solving programming problems. A qualitative methodology was employed. Both concept-driven coding and data-driven coding were applied. Two main issues emerged from the findings. Participating students had fragmented knowledge of the object-oriented approach and shortcomings regarding the implementation thereof, and they experienced problems with metacognitive control during all the steps of program development. Based on the findings the use of metacognitive critical control points (MCCPs) is proposed to be used as a mechanism to facilitate students in their programming efforts and to prevent loss of control during program development.

Keywords: Metacognition, problem solving, programming, thinking processes

Disciplines Computer Science, Education, Psychology

1. Introduction

This article reports on the transdisciplinary field of research by transcending the disciplinary borders of Computer Science, Education and Psychology. It shares the idea of integrating concepts and approaches from several disciplines to solve problems that cannot be dealt with in a singular discipline (Kroeze & Van Zyl, 2014, p. 3). The objective was to explore the role of metacognitive skills when students were solving object-oriented programming problems in a Computer Science course.

Metacognition refers to knowledge about our own thinking and *cognitive phenomena* (Flavell, 1979, p. 906), and has the ability to achieve deep and significant learning (Garrison & Akyol, 2015, p. 66). Metacognition involves the ability to think about our own mental activities, tasks and strategies, implement processes to direct and support cognitive thinking, and reflect on all actions performed, with the aim to enhance deep and significant learning (Flavell, 1979, p. 909; Sternberg & Sternberg, 2012, p. 234). Metacognitive thinking occurs as a cascade of related mental activities. It involves explicit planning, active control and critical evaluation of one's own cognitive processes, such as our own thoughts that engage in learning (Bergin, Reilly & Traynor, 2005, p. 82; Sternberg & Sternberg, 2012, p. 234; Titus & Annaraja, 2011, p. 14). Metacognition has two distinctive foci, namely metacognitive knowledge and metacognitive control of learning experiences (Flavell, 1979, p. 907-909; Miller & Geraci, 2011, p. 303). According to Titus and Annaraja (2011, p.15), metacognitive ability has a critical role in students' successful learning, especially in mental activities such as reasoning, comprehension and problem solving.

[1]. Dr. Marietjie Havenga, School for Natural Science and Technology Education, Faculty of Education Sciences, North-West University (Potchefstroom campus). Email: marietjie.havenga@nwu.ac.za

In addition, the use of metacognitive skills plays an important role in solving computer programming problems (Parham, Gugerty & Stevenson, 2010, p. 416); however, not much research has been done in this regard. Shaft (1995, p. 25-26) studied the use of metacognitive skills in program comprehension where professionals used the programming language COBOL. His research indicated that the use of metacognitive skills influences how well programmers understand a program. Shaft (1995, p. 25) asserts that programmers require the development of specific metacognitive heuristics to support them in comprehending programming tasks. Bergin et al. (2005, p. 82, 85) support Shaft's findings that students who performed well in introductory object-oriented programming, used more metacognitive strategies, such as planning, monitoring and regulation, than lower-performing students. Since the nature of object-oriented programming involves high-order thinking skills, such as reasoning, problem solving and abstract thinking, the use of metacognitive skills may support students in this regard. Consequently, the aim of the research reported here was to explore the role of metacognitive skills when students are solving object-oriented programming problems with reference to various steps involved in program development. The main research question was: What is the role of metacognitive skills when solving object-oriented programming problems in a Computer Science course?

The rest of the article is structured as follows: in Section 2, an overview is given of the conceptual-theoretical framework on which the empirical investigation was based. Section 3 is devoted to the empirical investigation and Section 4 to the results obtained. Section 5 discusses the findings.

2. Conceptual-theoretical framework

2.1 Metacognitive skills

Metacognition involves numerous skills that are fundamental in assisting students to manage their own understanding, thinking and learning (Falkner, Vivian & Falkner, 2014, p. 291; Garrison & Akyol, 2015, p. 66). The aim of metacognition is to direct students' thinking in such a way that they effectively control their mental activities especially when addressing real-life problems and complex tasks. A distinction is made between metacognitive knowledge and metacognitive control of experiences (Flavell, 1979, p. 907-909; Miller & Geraci, 2011, p. 303). Metacognitive knowledge involves knowledge of a person, knowledge of a task and knowledge of distinctive strategies to complete a task successfully (Flavell, 1979, p. 907), while metacognitive control refers to managerial processes to plan, monitor, reflect on and evaluate activities such as problem solving and critical thinking (Sternberg & Sternberg, 2012, p. 21; Titus & Annaraja, 2011, p. 14). Since the focus of the current study was mainly on metacognitive control, each of the managerial processes is outlined in more detail below.

Planning is associated with goal setting, reading of text and analysing of tasks to support understanding (Bergin et al., 2005, p. 82). Monitoring involves an individual's awareness of his/her state of cognitive activity, the skill to consider all detailed activities involved (e.g. problem solving), and the ability to assess his/her progress (Bergin et al., 2005, p. 85; Fletcher & Carruthers, 2012, p. 1366). Reflection as part of metacognition concerns that learners should reflect *in* action (while doing a task) and *on* action (after completing a task) (Schön, 1983). Lastly, evaluation is a metacognitive skill that determines the efficiency at which the task was performed, what students had to learn from the task, and whether the main goals had been achieved (Breed, Mentz & Van der Westhuizen, 2014, p. 53; Garrison & Akyol, 2015, p. 67).

The use of metacognitive skills is thus an endeavour undertaken to challenge one's own thinking, to monitor one's progress accurately, and to determine whether the aims had been accomplished. Metacognition is therefore a catalyst for change in cognitive thinking and behaviour to enhance meaningful and deep learning.

2.2 Application of metacognition in real-life problems

The importance of applying metacognitive skills as part of problem solving is emphasised by referring to two real-life examples. In industry, the application of metacognitive skills plays a significant role in food production, for example. Food safety requires strict specifications to manage possible hazards. To ensure food safety and trust of consumers, the hazard analysis critical control point system (HACCP) was developed with the aim to provide rigorous actions in identifying and preventing hazards that may occur in food production (Psomas & Kafetzopoulos, 2015, p.134). HACCP involves seven principles namely 1) conduct of hazard analysis, 2) identify critical control points, 3) establish critical limits for each critical control point, 4) establish critical control point monitoring requirements, 5) establish corrective actions, 6) establish procedures for ensuring the HACCP system is working as intended, and 7) establish record keeping procedures (Psomas & Kafetzopoulos, 2015, p. 134-135).

Another real-life problem is found where an air traffic controller needs to respond simultaneously to various inputs regarding aircraft to direct the controlling thereof. Solving real-life problems, such as those occurring during air traffic control, requires metacognitive factors (planning, monitoring, reflection, evaluation) to direct the reorganisation, priorities and management of air space as well as future aircraft movement and positions (Loft, Sanderson, Neal & Mooij, 2007, p. 390).

2.3 Metacognitive scaffolding of programming problems

Since the teaching of programming does not involve teaching a programming language only (Caruso, Hill, VanDeGrift & Simon, 2011, p. 498), additional skills are required to address misconceptions, increase students' understanding, address gaps in their knowledge, and to enable students in managing their own cognitive processes (Holliday, 2011, p. 2; Nussbaum, 2012, p. 116, 117; Parham et al., 2010, p. 416). Metacognitive scaffolding is therefore required during all steps of program development, such as understanding, planning and design, coding and testing. Each of these steps is outlined below.

Understanding results from rereading text, monitoring the reading activity and integrating knowledge structures into human memory to enable the processing and storage of information (Coiro, 2011, p. 108, 109; Gobet, Chassy & Bilalic, 2011, p. 187; Holliday, 2011, p. 2). Underlining of text, asking questions and addressing misconceptions may further support understanding of problems (Lee, Lim & Grabowski, 2010, p. 630; Nussbaum, 2012, p. 116).

Planning of solutions includes setting of goals, activating prior knowledge, breaking down a complex problem into manageable sections and consulting a strategy for problem solving (Bergin et al., 2005, p. 82; Sternberg & Sternberg, 2012, p. 448). In addition, the creation of hierarchies and concept mapping (Lee et al., 2010, p. 630) may support the design of object-oriented programs.

Program comprehension is part of program development, and is related to the understanding of programming code (Schulte, Clear, Taherkhani, Busjahn & Paterson, 2010, p. 65, 66). The purpose of program comprehension is to explore programmers' thinking processes when developing a computer program (Détienne, 1995, p. 164-166; Pennington, Lee & Rehder,

1995, p. 198-199). In this regard, the use of visualisation techniques, tools and debugging skills (Storey, 2006, p.187, 202) can be seen as metacognitive initiatives to support the understanding of computer programs. The interpretation of various solutions may further support program comprehension (Lee et al., 2010, p. 630). Metacognitive support is also required when developing test cases, determining program efficiency and assessing whether the goals had been achieved (Breed et al., 2014, p. 53; Caruso et al., 2011, p. 495).

In addition to the mentioned skills, the use of heuristics (mental shortcuts that lighten the cognitive load of making decisions) (Sternberg & Sternberg, 2012, p. 445) may further support students when solving problems, for example applying means-ends analysis (the problem solver analyses the problem with the final result in mind), working forward (to solve a problem from start to finish), and working backward (Sternberg & Sternberg, 2012, p. 493) (starting at the end and working backward, e.g. to determine whether the solution addressed all the programming requirements).

2.4 Object-oriented programming

When writing programs, students need to understand the programming approach (e.g. the object-oriented approach) they are using as this approach affects the way in which a program is written. The main building blocks of the object-oriented approach are 'objects' and 'classes'. An object comprises encapsulated data (e.g. *balance = 1000*) and methods (e.g. *getBalance*( )), that determine program behaviour (Farrell, 2008, p. 6; Satzinger, Jackson & Burd, 2004, p. 175). An object is based on a class for example a *client* is an object of the *Bank* class. Since novices experience limited understanding of the object-oriented (OO) approach (Ginat & Shmallo, 2013, p. 345; Sajaniemi, Kuittinen & Tikansalo, 2007, p. 2), students should be supported in this regard by applying, among others, metacognitive thinking. Havenga (2011, p. 96) is of the opinion that the more complex a programming problem is, the greater the need for metacognitive control, purposeful reflection and positive feedback.

To summarise this section, an overview was given regarding metacognitive thinking skills, the application thereof in real-life problems as well as metacognition's specific role in supporting object-oriented programming tasks. Consequently, the empirical research as described in the next section explored the role of metacognitive skills when students were solving object-oriented programming problems in a Computer Science course.

3. Empirical investigation

This section discusses the empirical research. The investigation was constructivist-based within an interpretivist approach (Hadjerrouit, 2005, p. 168,169; Kroeze, 2012, p. 9; Üredi, 2014, p. 228) to gain a deep understanding of the participants' metacognitive activities and programming experiences where they were actively involved in developing their own object-oriented programs. The aim was firstly to teach students metacognitive and problem-solving guidelines (Section 3.3.2) to support their thinking processes during programming and secondly to evaluate and gain an understanding how participants have implemented these mentioned guidelines as part of OOP.

3.1 Research design

A case study design (Gill, 2011, p. 10, 11) was employed to explore the role of metacognitive skills when students solve object-oriented programming. Merriam (1998, p. 27) emphasises that delimitation of the case being studied is "the defining characteristic" of a case study. In this research the case was bounded by the subject specification of students enrolling for

Computer Science with the aim of focusing on object-oriented programming (OOP) in this course. The purpose of this case design (Table 1) was to look at the succession of two programming tasks regarding students' detailed experiences during programming. This research was designed in such a way that there was a period of two weeks between the first and second programming assignment, in order to prevent other factors from having an influence over time. Table 1 displays the research design.

Table 1: Case study design

Introduction Week 1 and 2	**Assignment 1** Week 3	**Intervention** Week 4 and 5	**Assignment 2** Week 6
Introduction into OOP Explain OOP examples.	First programming assignment (individual work, see Section 3.3.1).	1) Explain the guidelines to support program development (see Section 3.3.2). 2) Discuss and apply the guidelines when developing an object-oriented program for the AB Bank (see Section 3.3.2) (group work, two students).	Second programming assignment (individual work, see Section 3.3.3). Conduct semi-structured interviews regarding students' experiences with the second assignment (see Section 3.4)

3.2 Participants

The participants were third-year students at a large South-African university taking Computer Science as one of their major subjects as part of their BEd degree. From the population of five students, four participated in this case study. No case study selection criteria were used as this was a small population of students. Participation was voluntary and all students completed informed consent forms. Ethical clearance was obtained from the university to conduct the study. Although the participants had previous experience regarding programming basics (e.g. iteration, selection), procedural programming, database skills and the use of arrays, they were novice OO programmers.

3.3 Programming assignments and the intervention

Two programming assignments as well as the intervention are discussed.

3.3.1 Programming Assignment 1

The students were required to plan, design and implement an object-oriented program that involved determining a total mark from a number of class tests. These marks were randomly generated. Although this problem can be solved in many ways, the rationale was to give the participants an easy object-oriented program where they were required to implement various methods as part of their introduction to OOP. Before starting to program, the students were required to write down their planning and design the solution. This assignment was done individually.

3.3.2 Intervention

The intervention was done in two separate parts (see Table 1). The first part comprised the teaching of metacognitive and problem-solving guidelines to support students' thinking activities and to direct their mental processes. The guidelines are summarised as follows:

- Read the programming problem and underline the main ideas and requirements. Write down the problem in your own words.

- Plan detailed steps and design a possible solution. Identify classes and draw a class diagram (additional diagrams may also be used).
- Code your planning in a programming language.
- Test the output. Indicate how well you have solved the problem.
- Continuously reflect and monitor all your activities. After completing the final solution (3.3.3), give yourself a mark out of 5, where 1=poor; 2=below average; 3=average; 4=good; and 5=excellent. Justify why you have given yourself this mark.

During the second part of the intervention (Table 1), the lecturer discussed a programming example and facilitated students in applying the above-mentioned problem-solving and reflective guidelines. This programming problem involved the following: write an object-oriented program for the AB Bank to create a new account for a client, enable the transfer and deposit of funds and close the account. The program is activated as soon as the PIN's correctness has been established. Participants worked together in groups of two during this assignment.

3.3.3 Programming Assignment 2

After the intervention, participants were given the following programming task: design an object-oriented program to display the amount due at a specific fuel pump after each vehicle had been filled-up. Click Stop to display the total amount after the morning shift. The pump number and amount were randomly generated. Participants had to plan and develop the program by using the metacognitive and problem-solving guidelines (see 3.3.2). Students had to complete this programming task individually during a practical lab session.

3.4 Data-collection activities

Data collection involved three aspects. Firstly, participants' planning, design, computer program and output (if output was obtained) of the first and second assignments were obtained. Secondly, their written problem-solving and reflective activities using the guidelines were collected (see 3.3.2). Thirdly, semi-structured interviews were held based on their experiences of the second assignment. The purpose of the semi-structured interviews was to clarify and elaborate on participants' thinking, metacognition and programming experiences. The interview questions were the following: *Explain the thinking processes you followed before starting to program; explain the processes used during programming and what difficulty you experienced; and reflect on your thoughts and activities regarding object-oriented programming*.

3.5 Analysis of qualitative data

Following the interviews, the students' reflections on their experiences were examined. Their programs were explored to check their understanding, approach, programming code, the solution and final program output (where applicable). Results from the interviews and written reflections were transcribed and manually analysed using both concept-driven (codes based on the literature) and data-driven coding (open coding) (Gibbs, 2010, p. 44-45). The main focus was on two matters, namely students' programming experiences and their use of metacognitive skills.

4. Results

Results are integrated and presented in this section. Participants' results from both assignments and the interviews are shown in Tables 2 to 5. The criteria indicated in these

tables were based on how participants understood, planned, designed and solved programming problems as well as on their reflections.

Table 2: Participant 1's (P1) results

Criteria	Assignment 1	Assignment 2
Problem comprehension	No in-depth analysis of the programming problem was made.	P1 wrote some requirements and identified some nouns and verbs.
Planning and program design	No planning was included. Presented three 'screen' buttons of the Graphical User Interface (GUI).	Planning was incomplete. Fragmented and incomplete design of the solution was indicated.
Program development	P1 programmed the solution without using the OO approach.	Programming of the class was incomplete.
Problems and errors	P1 was unsure how to program the solution when using OOP.	P1 made some programming errors. The class was not associated with the main application program.
Testing and self-assessment	Some output was obtained.	No output was obtained. Own mark allocation: none.
OOP experiences	The participant did not know how to create a new class and was unsure about the coding.	P1 did not use the new class as part of the application program.
Reflections and feedback	*I am not sure where to start and what to do. I have identified verbs and nouns. I find it difficult to plan. I never know what to use where and what to assign to the class. You can do it without using a new class ... it [OOP] is stupid.*	

Table 3: Participant 2's (P2) results

Criteria	Assignment 1	Assignment 2
Problem comprehension	P2 understood the problem.	P2 understood the problem and identified all nouns and verbs as part of problem analysis.
Planning and program design	Some ideas were written down.	He planned and designed the solution in detail and identified the required class.
Program development	P2 applied OOP. He could not solve the problem correctly.	P2 applied OOP and solved the problem correctly.
Problems and errors	Programming of the methods was incorrect.	The program worked after addressing some access violation problems.
Testing and self-assessment	Incorrect output was obtained.	Correct program output was obtained. His own mark allocation was 4 out of 5.
OOP experiences	He could not solve the problem correctly.	Successfully applied the object-oriented approach.
Reflections and feedback	*I read the problem and get a basic idea … thereafter I am planning the solution. I got some [execution] problems and did not know what the reason was for this … at the end the program worked in some manner, I think that I have solved the problem. I enjoy OOP … one advantage is better security.*	

Table 4: Participant 3's (P3) results

Criteria	Assignment 1	Assignment 2
Problem comprehension	P3 indicated understanding of the problem to some extent.	P3 comprehended the programming problem and identified some nouns and verbs.
Planning and program design	Although some ideas were written down, his planning was incomplete. He could not design the solution.	He did some planning how to solve the problem and designed the new class and application program.
Program development	P3 could not create an object.	P3's programming was incomplete.
Problems and errors	He experienced problems with program syntax as well as the programming of methods.	He used incomplete statements in the class.
Testing and self-assessment	No output was obtained.	No output was obtained. His own mark allocation was none.
OOP experiences	P3 did not know which methods to use and experienced an incomplete understanding of the OO approach.	He experienced problems with the programming syntax and semantics.
Reflections and feedback	*I understand the problem but do not know how to program this. I could not convert the planning into a program … I do not know how to do the programming. If you understand OOP you can use it effectively … I do not know what OOP is all about.*	

Table 5: Participant 4's (P4) results

Criteria	Assignment 1	Assignment 2
Problem comprehension	P4 indicated understanding of the programming problem.	He comprehended the problem and identified nouns and verbs.
Planning and program design	Some planning was included however he presented the 'screen', instead of designing the solution to the problem.	Some planning of the problem was included. The program design was incomplete.
Program development	He was unsure about the programming of methods.	P4 completed the class and application program.
Problems and errors	P4 was not sure when to use the specific methods.	He could not display the total amount after the morning shift.
Testing and self-assessment	No output was obtained.	Output was obtained but was incorrect. His own mark allocation was 4.
OOP experiences	He experienced difficulty in comprehending the OO approach.	He still experienced difficulty in comprehending OOP.
Reflections and feedback	*I never plan in detail … I am thinking in a programming language when I plan. When starting with the programming, I have made changes [changed the initial planning]. I am not sure what precisely should happen. It is sometimes still difficult to comprehend the OOP concept.*	

5. Discussion of the findings

This section addresses the research question: What is the role of metacognitive skills when solving object-oriented programming problems in a Computer Science course?

The approach followed in this section is to discuss participants' individual experiences regarding the first and second assignments. This is followed by a discussion of all students' experiences and the overall findings. The discussion is elaborated by integrating results from the interviews.

5.1 Findings of individual participants

Findings of Participant 1: P1 experienced the following problems: an inability to understand and apply the object-oriented approach and incompetence regarding the use of metacognitive control in terms of planning, monitoring and evaluation (see 2.1, 2.3, Table 2). P1 preferred to program without using the OO approach. He made some syntax and semantic errors and experienced problems in programming the methods. P1 realised that he did not clearly understand the problem requirements and made no in-depth analysis of both programming problems (Table 2). His attempts were fragmented and did not proceed towards solving both programming problems. With reference to metacognition, P1 reflected: *I am not sure where to start and what to do,* however his reflection was not followed by active monitoring to address the problems he experienced. Participant 1's planning was incomplete, he did not monitor his actions and did not evaluate and self-assess his efforts. As a result he was not able to plan, design and code the programs and solve the two problems. Bergin et al. (2005, p. 82) emphasise explicit planning and active control to direct one's own cognitive processes during program development.

Findings of Participant 2: Results from Table 3 indicate that Participant 2 understood both problems and the requirements. He made an analysis of the second programming assignment and was able to plan and design the solution after applying the problem-solving and reflective guidelines (see Section 3.3.2). Although he encountered minor challenges in solving the first assignment, it was evident that P2 experienced a higher level of overall understanding in the second assignment than the first. He monitored the programming process, reflected on his programming efforts and made the required corrections: *I got some access violations … at the end the program worked.* P2 obtained incorrect output from the first programming task; however he solved the second problem correctly and his own mark allocation was 4 out of 5.

Findings of Participant 3: Participant 3 understood both programming assignments to some extent; however the planning and design of both solutions were incomplete. The main obstacles were an incomplete understanding of the object-oriented approach as well as problems with program comprehension (see Section 2.3). *I understand the problem but do not know how to program this* (Table 4). This student struggled with the coding, the programming syntax, semantics and OO constructs. Ginat and Shmallo (2013, p. 345) concur that novices in their study experienced limited understanding of the OO paradigm. In addition, P3 experienced problems with various programming statements and methods. He was not able to complete both programming tasks and could not obtain output. Participant 3 did some planning, however he did not monitor his progress, follow up or correct the errors. Results from Table 4 indicate that there was nearly no progress towards developing the second assignment.

Findings of Participant 4: P4 was able to understand both assignments although he had problems in identifying classes in the first programming task (Table 5). He outlined the GUI

instead of giving a detailed plan regarding how to solve the problem. P4 mentioned *I never plan in detail* and this was an obstacle in the process of solving the problem correctly. Although he obtained some output, P4 did not monitor and correct the output errors. Furthermore, he was unsure how to interpret the object-oriented approach. Regarding his own assessment, he indicated a mark of 4 out of 5.

5.2 Participants' overall experiences

Results from both assignments as well as the interviews and reflections (Tables 2 to 5) indicate that the participating students experienced the following problems: 1) fragmented knowledge and misconceptions of the object-oriented approach, shortcomings regarding the implementation of OOP, and 2) inefficient and inadequate metacognitive control during all steps of program development.

Firstly, participants (P1, P3 and P4 to some extent) had fragmented knowledge and shortcomings in understanding and applying the object-oriented approach. They did not understand the problem requirements clearly, could not analyse the problems in depth and made various syntax and semantic errors (P1 and P3). Except for P2, analyses of the remaining students' programs revealed that they struggled to combine code and relevant constructs to produce an object-oriented program that executed correctly. Since Participant 4 was not used to planning in detail, his design of the solution in the second assignment was incomplete, and as a result, his program output was not correct. Previous research (Sajaniemi et al., 2007, p. 2) also indicated students' challenges in applying OOP with specific reference to a change from procedural to object-oriented programming, and a limited understanding of the OO paradigm and concepts (Ginat & Shmallo, 2013, p. 345; Govender, 2010, p. 14, 15).

Secondly, participating students experienced difficulty in applying metacognitive skills. Distinctive skills, such as detailed planning, the ability to monitor their own progress (Bergin et al., 2005, p. 85; Fletcher & Carruthers, 2012, p. 1366) and accurate judgement to determine whether the goals had been achieved (Garrison & Akyol, 2015, p. 67), were clearly absent in most participants. Participant 2 mentioned the use of reflective skills when his program worked after addressing some execution problems (Table 3).

Regardless of the fact that P1, P3 and P4 (to some extent) had knowledge about the problem-solving and reflective guidelines (Section 3.3.2), they did not implement these adequately and were not able to proceed towards solving the OOP problems. These students did not control their own learning processes as referenced by Flavell (1979, p. 907-909) and Miller and Geraci (2011, p. 303). Negative reflection, e.g. *I am not sure where to start and what to do* (P1, Table 2) without positive feedback and corrective activities will be an obstacle in solving programming problems successfully (Havenga, 2011, p. 95, 96). It seems that when participants struggled and got stuck, they experienced an accumulation of problems in subsequent steps as indicated by P1's reflections: *I am not sure where to start and what to do. I never know what to use where and what to assign to the class.* Falkner et al. (2014, p. 291) assert that, without a fundamental level of metacognition, students cannot direct their knowledge in a constructive manner. Deliberate integration of metacognitive skills during program development is therefore crucial and students need to be directed in this regard, as mentioned by Titus and Annaraja (2011, p.15) (Section 1).

5.3 Application of metacognitive critical control points

Although participating students applied the metacognitive and problem-solving guidelines (Section 3.3.2) to some extent, the results were unsatisfactory (Sections 4, 5.1 and 5.2).

Further interventions are therefore proposed, though not tested, to support programming students in this regard.

In Section 2.2 the application of metacognition in real life is outlined with reference to its use as part of a HACCP system to provide rigorous actions in identifying and preventing food hazards (Psomas & Kafetzopoulos, 2015, p.134). Accordingly, the author of this paper applies these principles and postulates the use of metacognitive critical control points (MCCPs) as a metacognitive mechanism to enable students in supporting and managing their own thinking processes during all steps of program development. The aim is to teach programming students how to direct their own thinking processes, to enable them in proceeding towards successful completion of a task, to improve their overall metacognitive managerial skills, and to prevent loss of control (where a student is no longer being able to manage his or her thinking when trying to solve a problem). The detailed activities using MCCPs are the following:

1. Each student should conduct his/her *own* reflective analysis regarding previous programming experience(s) since he/she should think about his/her *own* cognition the application of MCCPs will therefore differ among students.
2. A student should identify metacognitive critical control points for each step of program development to address challenges, gaps and problems he/she had experienced previously (as mentioned in the above activity).
3. The next step is to establish specific requirement(s) for *each* critical control point as mentioned in point 2.
4. The student should apply monitoring and reflective procedures that allow one to address the requirements for each critical control point.
5. Repeat steps 2 to 4 (shaded in Table 6) to ensure that all the critical points (mentioned in point 2), were addressed.
6. Lastly the student should evaluate whether all metacognitive critical control points have been met to support the successful completion of the programming task. A record should be kept of these reflective evaluations for future reference.

The application of MCCPs and relevant literature (Section 2) in each step of program development are used with reference to the problems that P1 experienced. Note that the analysis following here is mainly focused on this participant's first problem: *I am not sure where to start and what to do* (number 1, Table 6) since it is not possible to outline each of the problems in the same way due to problems with space.

Table 6 shows the integration of MCCPs during program development with specific reference to the problems and challenges that Participant 1 experienced (Table 2). Regarding the first step of MCCPs, analysis indicates that P1 was not sure where to start and what to do. In the next step MCCPs were identified to address this problem, namely understanding the problem and determining what to do to direct the problem-solving process. Step 3 establishes requirements to address the first control point. The fourth step involves detailed monitoring and reflective actions to address the requirements for a specific control point as mentioned in the previous step.

Steps 2 to 4 are repeated to ensure that all critical points have been met. When evaluating the programming solution in the final step, a student is required to understand, test and reflect on *all* previous steps and processes to ensure that the required MCCPs had been addressed and the problem had been solved.

Table 6: An example of how to apply MCCPs in supporting P1

MCCP activities	The application of MCCPs in program development
1. Conduct your own reflective analysis regarding a previous programming experience	Conduct your own reflective analysis: -***I am not sure where to start and what to do*** (first challenge of this participant) -*I find it difficult to plan* (2) -*I never know what to use where and what to assign to the class* (3) -*You can do it without using a new class [OOP]* (4) (Table 2)
2. Identify metacognitive critical control points for each step of program development (see Section 2.3)	Identify critical control points to: -understand the problem and determine what to do (no **1**) -plan and design the solution (2) -enhance program comprehension, direct coding of the program (3) -evaluate the solution and your understanding of OOP(4)
3. Establish specific requirements for each critical control point	Establish requirements for **no 1** to: -understand the problem description -specify detailed actions, heuristics and strategies on what to do.
4. Apply monitoring and reflective procedures that allow you to address the requirements for **each** critical control point	Establish and apply monitoring and reflective procedures to enhance understanding of **no 1**: -pay focused attention, reread the problem, comprehend what is required (Bergin et al., 2005, p. 82; Coiro, 2011, p. 108) -determine the intended meaning of the problem and reflect towards the clarity thereof -interpret and write the programming problem in your own words -make connections and recall previous knowledge -address any misconceptions (Lee et al., 2010, p. 630; Nussbaum, 2012, p. 116) -ensure deep understanding of the problem, what exactly is required in terms of program development and reflect on your thinking -comprehend additional OOP problems and solutions -make decisions regarding actions, heuristics and strategies (Sternberg & Sternberg, 2012, p. 448), e.g. underline all nouns and verbs as an indication of 'things' (objects) and 'events' (methods) -reflect *in* action (Schön, 1983), when solving a programming problem
5. Repeat steps 2 - 4	
6. Evaluate whether all metacognitive critical control points have been met to support the successful completion of a programming task.	Evaluate whether all metacognitive critical control points have been met: -determine whether the requirements of each critical control point were addressed and reflect *on* action (Schön, 1983) -determine whether the main goals were achieved and if you solved the problem effectively (Garrison & Akyol, 2015, p. 67) -ask yourself questions about the correctness and efficiency of your programming solution -discuss your solution with peers and compare various solutions -keep record of your reflective evaluations for future reference

6. Conclusion

This investigation was aimed at determining whether students apply metacognitive skills when solving object-oriented programming problems. Results indicate that the participating students had difficulty in understanding object-oriented problems and they displayed shortcomings regarding the implementation thereof. In addition, most participants experienced problems with metacognitive control during all steps of program development.

The author postulates the use of metacognitive critical control points (MCCPs) as a mechanism to manage and facilitate all thinking involved in object-oriented program development. The aim is to teach students how to manage their own thinking processes, to enable them in proceeding towards successful completion of a task, and to prevent loss of control.

It is essential that lecturers support students in the application of MCCPs during programming to direct their own thinking processes and activities. Some limitations of the current study involved a small population, and participants had only two formal programming assignments to complete. Initial findings should therefore be tested and verified using a larger cohort of students. Future research could focus on the effectiveness of applying MCCPs in problem-solving tasks.

Acknowledgement

The author would like to acknowledge the Dean of our Faculty for providing funds, as well as the students who participated in this research.

Bibliography

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. *International Computing Education Research.* Seattle, WA (October 1-2).

Breed, B., Mentz, E., & Van der Westhuizen, G. (2014). A metacognitive approach to pair programming: Influence on metacognitive awareness. Retrieved on 4 March 2015 from http://www.investigacion-psicopedagogica.org/revista/articulos/32/english/Art_32_886.pdf

Caruso, T., Hill, N., VanDeGrift, T., & Simon, B. (2011). Experience report: Getting novice programmers to THINK about improving their software development process. Retrieved on 9 March 2015 from http://dl.acm.org/citation.cfm?id=1953307&dl=ACM&coll=DL&CFID=487923427&CFTOKEN=59087041

Coiro, J. (2011). Talking about reading as thinking: Modeling the hidden complexities of online reading comprehension. *Theory into Practice,* 50, 107-115.

Détienne, F. (1995). Design strategies and knowledge in object-oriented programming: Effects of experience. *Human-Computer Interaction,* 10, 129-169.

Falkner, K., Vivian, R., & Falkner, N.J.G. (2014). Identifying computer science self-regulated learning strategies. Retrieved on 23 February 2015 from http://dl.acm.org/citation.cfm?id=2591715

Farrell, J. (2008). *An object-oriented approach to programming logic and design.* (2nd ed). Boston, MA: Thomson Course Technology.

Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive-developmental inquiry. *American Psychologist*, 34(10), 906-911.

Fletcher, L., & Carruthers, P. (2012). Metacognition and reasoning. *Philosophical Transactions of the Royal Society Biological Sciences,* 367, 1366-1378.

Garrison, D.R., & Akyol, Z. (2015). Toward the development of a metacognition construct for communities of inquiry. *Internet and Higher Education,* 24, 66-71.

Gibbs, G. (2010). *Analyzing qualitative data*. Los Angeles, CA: SAGE.

Gill, T.G. (2011). *Informing with the case method: A guide to case method research, writing, and facilitation.* Santa Rosa, CA: Informing Science Press.

Ginat, D., & Shmallo, R. (2013). Constructive use of errors in teaching CS1. SIGCSE, Denver, CO (March 6-9).

Gobet, F., Chassy, P., & Bilalic, M. (2011). *Foundations of cognitive psychology*. London: McGraw-Hill.

Govender, I. (2010). From procedural to object-oriented programming (OOP): An exploratory study of teachers' performance. *South African Computer Journal,* 46, 14-23.

Hadjerrouit, S. (2005). Object-oriented software development education: A constructivist framework. *Informatics in Education,* 4(2), 167-192.

Havenga, M. (2011). Problem-solving processes in computer programming: a case study. SACLA, Ballito, KwaZulu-Natal, SA (July 6-8).

Holliday, M.A. (2011). Reading strategies and student comprehension in an Internet ethics course. ASEE/IEEE Frontiers in Education Conference, Rapid City, SD (October 12-15).

Kroeze, J.H. (2012). Interpretivism in IS – a postmodernist (or postpositivist?) knowledge theory. Proceedings of the 18th Americas Conference on Information Systems (AMCIS 2012 Proceedings), Seattle, Washington, USA (August 9-11), 2012, Paper 7, ISBN 978-0-615-66346-3. Available: http://aisel.aisnet.org/amcis2012/proceedings/PerspectivesIS/7 (July 29, 2012) or http://hdl.handle.net/10500/6983.

Kroeze, J.H., & Van Zyl, I. (2014). Transdisciplinarity in Information Systems: Extended Reflections. Twentieth Americas Conference on Information Systems (AMCIS 2014 proceedings), pp. 1-10. Savannah, Georgia (August 7-9).

Lee, H.W., Lim, K.Y., & Grabowski, B.L. (2010). Improving self-regulation, learning strategy use, and achievement with metacognitive feedback. *Educational Technology Research and Development,* 58, 629-648.

Loft, S., Sanderson, P., Neal, A., & Mooij, M. (2007). Modeling and predicting mental workload in En route air traffic control: Critical review and broader implications. *Human Factors,* 49(3), 376-399.

Merriam, S.B. (1998). *Qualitative research and case study applications in education*. San Francisco, CA: Jossey-Bass.

Miller, T.M., & Geraci, L. (2011). Training metacognition in the classroom: The influence of incentives and feedback on exam predictions. *Metacognition Learning,* 6, 303-314.

Nussbaum, E.M. (2012). Argumentation and student-centered learning environments. In D. Jonassen & S. Land (Eds.), *Theoretical Foundations of Learning Environments* (2nd ed., pp. 114-141). New York, NY: Routledge.

Parham, J., Gugerty, L., & Stevenson, D.E. (2010). Empirical evidence for the existence and uses of metacognition in computer science problem solving. *Special Interest Group on Computer Science Education Bulletin*, 416-420.

Pennington, N., Lee, A.Y., & Rehder, B. (1995). Cognitive activities and levels of abstraction in procedural and object-oriented design. *Human-Computer Interaction,* 10, 171-226.

Psomas, E.L., & Kafetzopoulos, D.P. (2015). HACCP effectiveness between ISO 22000 certified and non-certified dairy companies. *Food Control,* 53, 134-139.

Sajaniemi, J., Kuittinen, M., & Tikansalo, T. (2007). A study of the development of students' visualizations of program state during an elementary object-oriented programming course. *ICER*, Atlanta, GA (September 15-16).

Satzinger, J.W., Jackson, R.B., & Burd, S.D. (2004). *Systems analysis and design in a changing world.* (3rd ed.). Boston, MA: Thomson Course Technology.

Schön, D.A. (1983). *The reflective practitioner: how professionals think in action* London: Temple Smith.

Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J.H. (2010). An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports* (ITiCSE-WGR '10), Alison Clear and Lori Russell Dag (Eds.). ACM, New York, NY, USA, 65-86. DOI=10.1145/1971681.1971687 http://doi.acm.org/10.1145/1971681.1971687

Shaft, T.M. (1995). Helping programmers understand computer programs: The use of metacognition. *Data Base Advances,* 26(4), 25-46.

Sternberg, R.J., & Sternberg, K. (2012). *Cognition*. (6th ed.). US: Wadsworth Cengage Learning.

Storey, M.-A. (2006). Theories, tools and research methods in program comprehension: Past, present and future. *Software Quality Journal,* 14, 187-208.

Titus, S.V., & Annaraja, P. (2011). Teaching competency of secondary teacher education students in relation to their metacognition. *International Journal on New Trends in Education and their Implications,* 2(3), 14-22.

Üredi, L. (2014). Analyzing the classroom teachers' levels of creating a constructivist learning environments in terms of various variables: A Mersin case. *Educational Research and Reviews*, 9(8), 227-236.