Thinking processes used by high-performing students in a computer programming task

M HAVENGA,[1] R DE VILLIERS[2] AND E MENTZ[3]

Abstract

Computer programmers must be able to understand programming source code and write programs that execute complex tasks to solve real-world problems. This article is a trans-disciplinary study at the intersection of computer programming, education and psychology. It outlines the role of mental processes in the process of programming and indicates how successful thinking processes can support computer science students in writing correct and well-defined programs. A mixed methods approach was used to better understand the thinking activities and programming processes of participating students. Data collection involved both computer programs and students' reflective thinking processes recorded in their journals. This enabled analysis of psychological dimensions of participants' thinking processes and their problem-solving activities as they considered a programming problem. Findings indicate that the cognitive, reflective and psychological processes used by high-performing programmers contributed to their success in solving a complex programming problem. Based on the thinking processes of high performers, we propose a model of integrated thinking processes, which can support computer programming students.

Keywords: Computer programming, education, mixed methods research, thinking processes.

Disciplines: Computer programming, education, psychology.

1. Introduction

The research described in this article lies at the intersection of computer programming, education and psychology. It is an inter-disciplinary study that investigates how the thinking processes and strategies used by high-performing student programmers foster the development of correct computer programs.

The primary aim of a computer programmer is to write correct, high quality computer programs that solve real-world problems effectively and efficiently. A computer program is a human artefact comprising language-specific rules, formulated in the so-called source code of a particular programming language. This coded language communicates functionality that the computer hardware can 'understand'. By 'syntax' of a program, we refer to expressions, statements and program units, whereas 'semantics' refer to the meaning of those expressions, statements and program units (Sebesta, 2008). Syntax and semantics are closely related, and the specific way in which programming

[1]. Faculty of Education Sciences, North-West University, Private Bag X6001, Potchefstroom, +27 18 299 4281, marietjie.havenga@nwu.ac.za

[2]. School of Computing, University of South Africa, P O Box 392, Unisa, 0003, South Africa, +27 12 429 6559, Dvillmr@unisa.ac.za.

[3]. Faculty of Education Sciences, North-West University, Private Bag X6001, Potchefstroom, +27 18 299 4281, elsa.mentz@nwu.ac.za +27 18 299 4238(f).

statements, expressions and constructs are organised and combined, reflects the meaning and purpose of a program. A programmer's approach, knowledge and thinking shapes the structure, logic, and flow of the program. Student programmers undergo the process of learning detailed programming knowledge and skills. Some are particularly competent in applying a variety of skills, forms of knowledge, and thinking processes in their programming. In order to understand the mental activities and strategies of such students, we investigated their approaches to a particular programming task, along with their underlying thoughts and reasoning, documented during the programming process. In previous papers we discussed the differences between successful and unsuccessful programming students (Havenga, Mentz & De Villiers, 2008) and suggested how average programming students could improve on their performance (Havenga, De Villiers & Mentz, 2010). These publications came from different perspectives, while the aim of the present research was specifically to determine in what ways the thinking processes used by *highly competent* student programmers, support their programming performance. The research question addressed in this study was:

> *How can the specific thinking processes used by high-performing computer science students, support them in writing correct and well-defined programs?*

To answer this, we overview relevant literature in Section 2, and set out the research design and methodology in Section 3. Sections 4 and 5 describe the respective qualitative and quantitative methods used, while Section 6 consolidates the results and presents a diagrammatic representation of integrated thinking processes in the context of programming. Section 7 briefly concludes the study.

2. Literature overview

This paper describes an integration of the domains of computer programming, psychology and education by inter-relating the thinking processes of computer science students with the actual outcomes of their computer programming tasks. To this end, we address relevant literature on, first, the role of the human mind in the activities required in the programming process and, second, the particular forms of knowledge and regulatory mental strategies applied while undertaking programming.

2.1 Involvement of the human mind in programming

Programmers must be able to understand the syntax and semantics of source code, and write programs that execute complex tasks to solve the problem in hand. The way in which this is done, is based on the programmer's underlying thinking and decision-making processes. Various mental activities are involved in solving programming problems. According to Parnin (2010), the memory used to program *syntax*, is retained by means of abstracted perceptual patterns or visual sketches. An example of this is the use of indented and highlighted text to display the iterative patterns of a 'for' loop construct, which is a programming mechanism used to repeat a statement or set of statements. The *semantic* meanings of programming statements are retained by using auto-associative support from the hippocampal formation (Parnin, 2010). The hippocampus in the brain is involved in the encoding and retrieval of memory processes (Chen, Chuah, Sim & Chee, 2010).

Working memory refers to the temporary storage and maintenance of information required in various types of cognitive and complex tasks (Klingberg, 2010; Unsworth, Redick, Heitz, Broadway & Engle, 2009). Working memory is involved in high-level thinking processes such as comprehension, reasoning, problem solving and the use of language (Unsworth et al., 2009). It integrates acoustic and visual information, organises it meaningfully, and links new information to existing knowledge in long-term memory (Sternberg, 2006). However, in order to complete complex tasks successfully, information must be actively maintained in working memory. Effective teaching of mental strategies and extended training might support maintenance of working memory by improving its capacity, resulting in associated change in brain activity in various sections (Klingberg, 2010).

2.2 Knowledge and strategies used in the programming process

Programmers need to maintain and recall different types of *knowledge* as they proceed through the process of computer programming. In order to write programs that execute effectively and that produce correct output, declarative, procedural and metacognitive knowledge are required (Ismail, Ngah & Umar, 2010). *Declarative knowledge* is the knowledge of facts that can be stated (Sternberg, 2006), for example, knowledge of the syntax of the 'for' construct mentioned in the previous section. *Procedural knowledge* refers to the implementation of knowledge, that is, knowing how and when to conduct certain processes or activities (Gunter, Estes & Mintz, 2010), such as the execution of a particular segment of programming code. *Metacognitive knowledge* relates to an individual's explicit knowledge about his/her own cognitive learning processes, strengths and weaknesses (Gravill, Compeau & Marcolin, 2002; Gunter et al., 2010), for example, knowledge regarding how to use particular strategies in the process of solving programming problems. The effective use of all three forms of knowledge, namely: declarative, procedural and metacognitive knowledge, directs a programmer's mental processes and can enhance the use of the associated cognitive, metacognitive and problem-solving skills and strategies, which are addressed in the next paragraphs.

In order to apply assimilated knowledge effectively, various supportive *strategies and activities* can be used. *Cognitive activities* refer to the mental processes used in the acquisition, storage, transformation and application of appropriate content knowledge (Sternberg, 2006). The relative complexity of the content can be managed cognitively by using different types of knowledge in different ways. Various schemes exist to represent such dimensions. Some schemes are taxonomical, while others emphasise varying categories (Webb, 2002). We used the classic Bloom's Taxonomy, which has been applied in Computer Science Education (Börstler & Schulte, 2005), to analyse the different types of knowledge used and applied by programming students. This taxonomy presents six levels of cognitive activities: knowledge, comprehension, application, analysis, synthesis and evaluation (Bloom, Krathwohl & Masia, 1973).

Gravill et al. (2002) indicate that metacognitive activities play a critical role in successful programming. *Metacognitive strategi*es include planning, monitoring and self-regulation that affect reflection and memory performance (Bergin, Reilly, & Traynor, 2005; Flavell, 1979). The use or non-use of these practical activities, impacts both on the programming process and the resulting product. Furthermore, programmers use different types of *problem-solving activities* when initially considering a programming problem, examples being the top-down, bottom-up and integrated strategies. The top-down approach addresses the 'big picture' and subsequently decomposes it into smaller subproblems (Storey, 2006). By contrast, when using the bottom-up strategy, programmers focus initially on details of the individual parts that are combined into higher-level abstractions (Storey, 2006). The integrated strategy combines top-down and bottom-up strategies in different levels of abstraction.

In their interactive learning model, Tennyson and Nielsen (1998) expound the relationship between the cognitive and affective domains with regard to interaction of content knowledge and cognitive strategies for higher-order thinking processes such as problem solving, creativity, decision making and trouble shooting.

The concepts mentioned in this section are examples of knowledge, skills and strategies that might support students in computer programming. In the sections following, we describe empirical research relating to high-performing students who implemented such mental activities while undertaking a programming task.

3. Research paradigm and methodology

3.1 Research approach

The research described in this paper employs a mixed approach (Creswell, 2008; Creswell, 2009), whereby qualitative and quantitative methods, associated with interpretivism and postpositivism respectively, are applied in tandem. We used both interpretivist and postpositivist approaches, so as to holistically analyse programming students' interpretations of the problem, as well as to grade their programming performance (Havenga, 2008) and to determine why some students succeed.

The interpretivist paradigm relates to knowledge and deep insight that are intentionally obtained by the interpretation of constructs through the lived experience of human beings (De Villiers, 2005; Klein & Myers, 1999). This approach uses mainly qualitative methods that rely on non-statistical techniques of data collection and analysis. We applied grounded theory (Glaser & Strauss, 1967; Strauss & Corbin, 1998; Creswell, 2008) as a method (Matavire & Brown, 2008) to focus on the psychological dimensions of the students' thinking processes and problem-solving activities as they considered a programming problem.

Postpositivism is a quantitative-based approach that involves quantitative and, on occasions, qualitative methods (Onwuegbuzie, Johnson & Collins, 2009). Based on postpositivism, we assigned quantitative scores to defined criteria, including measurement of some qualitative aspects, based on the use of the grounded theory method (GTM). As indicated in Section 3.2, the participants in this study were the high-performing students from two cohorts of programming students.

Figure 1 shows a diagram of a mixed methods approach, in which a variety of processes are applied. In the general context, these include data collection and analysis, structural organisation of qualitative data, and the systematic investigation of quantitative data. In the specific context of the present study, it comprises firstly the collection and analysis of students' recorded thinking processes and their programs; the organisation and categorisation of qualitative data; the identification of themes and patterns that emerge and the subsequent generation of theory. These processes are displayed in an upward progression in the left quadrilateral of Figure 1. The generation of theory is an abstract, inductive process (Creswell, 2008; Gibbs, 2010), progressing from detailed thinking processes and programs to general themes and the induction of a theory—hence the vertical broadening of the quadrilateral towards its culmination at the top.

Secondly, we analysed empirical data by using quantitative methods such as descriptive and inferential statistics. The upward broadening of the quadrilateral on the right indicates the generalisation of results. Finally the two sets of results were integrated for interpretation as shown at the top of Figure 1.

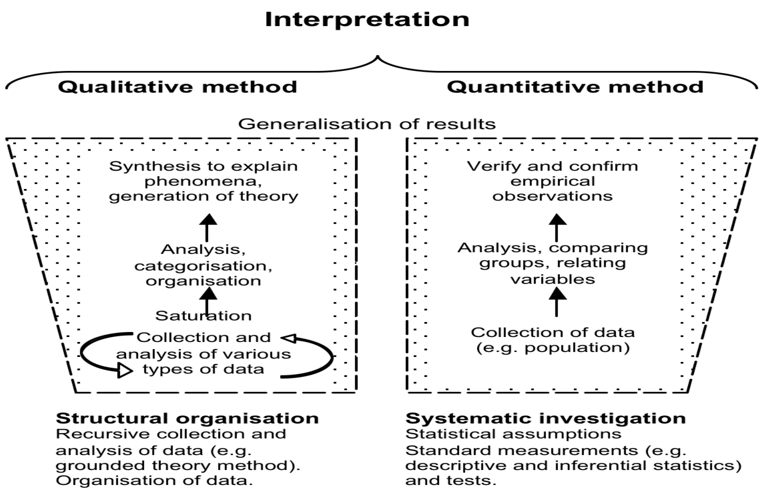
Interpretation
Qualitative method
Quantitative method
Generalisation of results
Synthesis to explain phenomena, generation of theory
Analysis, categorisation, organisation
Saturation
Collection and analysis of various types of data
Verify and confirm empirical observations
Analysis, comparing groups, relating variables
Collection of data (e.g. population)
Structural organisation
Recursive collection and analysis of data (e.g. grounded theory method). Organisation of data.
Systematic investigation
Statistical assumptions
Standard measurements (e.g. descriptive and inferential statistics) and tests.

Figure 1: Application of a mixed methods approach

The rationale for integrating qualitative and quantitative methods in this research is to emphasise completeness or the "notion that the researcher can bring together a more comprehensive account of the area of enquiry" (Bryman, 2006, p. 106). It implies that qualitative and quantitative methods, as separated instruments, do not completely explain why some programmers are successful.

3.2 Participants and the programming problem

Participants

The participants in the overarching research venture, which was conducted over a period of two years, were third-year, exit-level students taking Computer Science or Information Technology as a major subject. The participants (*n*=48) came from two cohorts; the first cohort consisted of 11 BEd and 17 BSc third-year students and the second cohort comprised three BEd and 17 BSc third-year students. The group of participants was not a sample; it was the population of students doing computer programming as part of their degree studies. Participation was entirely voluntary and all students completed informed consent forms. The present study focuses not on the performance of the full set of 48 participants, but on 11 high performers, so-called 'successful programmers', as explained in Section 5.1.

Programming problem

The programming problem involved designing and writing a computer program to perform complex calculations with dates, as well as a test program to check the output of the 'date' program. The students had two weeks to complete the programming task, as it was not part of their class assignments. An initial task framework, in the form of an open-ended programming question, was given to guide the data collection process. At the very least, participants were expected to write a computer program called *Date class* to determine which years are leap years and to calculate the difference in days between any two dates in the range 1 January 1800 to any later date up till the present. They were also required to write a second program called *Test class* to determine whether the output of the *Date class* was correct. These programs could be written in either the Delphi or Java programming language, both of which are object-oriented languages.

During the process of interpreting the question and writing these object-oriented programs (OOP), participants had to reflect on their experiences of programming and write descriptive textual documents to record their associated thinking and problem-solving processes. This reflective journaling exercise provided descriptive data to be analysed by qualitative processes. To ensure uniformity in the structure and content of the text documents, the task description gave precise instructions as to what aspects should be covered and how they should be set out (Havenga, 2008). Data collection therefore involved both the computer programs and the written thinking processes

4. Qualitative data collection and analysis

4.1 Collection and analysis processes

The grounded theory method was applied to analyse students' thinking processes recorded in their journals. GTM involves a sequence of actions such as coding; theoretical sampling and constant comparative analysis; defining and refining properties and categories; and identifying their relevant contexts. In this way, recursive collection and analysis of data proceed towards the identification of themes and the generation of theory regarding participants' thinking processes. The cyclic left and right open arrows in the qualitative section of Figure 1 represent this recursive process.

Atlas.ti (Muhr, 2004) was selected as a powerful software knowledge workbench, to optimise the coding and analysis processes. Within *Atlas.ti* the textual information from each individual's thinking processes was assigned to a primary document. All primary documents were integrated into a single hermeneutic unit. Specific codes were awarded to selected sections of text in each primary document, to represent explicit ideas or meanings. Saturation of data did not occur until near the very end. The codes in the hermeneutic unit were organised into possible groups of related codes or coded 'families'.

Three months later, further analyses were repeated on the same data, to refine the coded families and to increase trustworthiness of the emergent themes and patterns.

4.2 Qualitative findings and the development of a grounded theory

Themes that emerged from the data

As part of the inductive generation of emergent theory, the following five themes emerged, representing characteristics of the students' programming processes:

- Use of cognitive knowledge, skills and strategies;
- Use of metacognitive thinking processes;
- Application of problem-solving strategies;
- Handling of errors and problems that occurred in the development of the computer programs; and
- Additional forms of support used.

With regard to the first three, the strategies concerned were used to a greater or lesser extent by different participants. This study, however, investigates the performances of the most successful students, so as to determine whether successful thought processes and reflection support students in similarly attaining success in their programming activities. The formal definition of a 'successful programmer' is related to quantitative scores and is given in Section 5.1 on quantitative data analysis. We now provide examples of occurrences of the five themes as indicated in the journaling of the successful programmers.

Clear evidence of *cognitive activities* emerged from analysis of the data of these high-performing students. A list follows of quotations from their textual thinking processes, linked to associated levels of Bloom's taxonomy (in parentheses). [P29] refers to Participant 29, etc.

> *I begin with the class and constructor [P15] (Knowledge);*
>
> *I determined the days of each month [P12] (Comprehension);*
>
> *I think about the screen layout [P29] (Application);*
>
> *Determine the difference between two dates [P42] (Analysis);*
>
> *Subtract 1800 from [the] date and also determine leap years [P38] (Synthesis);*
>
> *The program is now working 100% [P40] (Evaluation).*

Knowledge is required about the class construct [P15] and *comprehension* skills are used to determine the number of days in a relevant month [P12]. The examples above illustrate that the participants who are quoted, recalled facts and interpreted the programming problem. P29 used *application* skills when he considered, and later designed, a user-friendly interactive screen layout to determine leap years, and when he decided on the associated input and output. Whilst *analysing* the programming problem, decisions should be made about the time range of a given set of two dates [P42]. In writing a computer program, there are usually several different ways to operationalise the requirements. During *synthesis*, therefore, participants applied appropriate formulae, then used a variety of programming constructs and code syntax to write the program, for example, methods to determine the leap years [P38]. *Evaluation* should be used to determine whether the complete program works, as did P40, who self-reviewed, noting that he made the necessary changes and that his program executed correctly thereafter.

The use of *metacognitive activities* was clearly evident in most participants' thinking processes. Prior knowledge acquired from previous programming tasks can ease planning for a new task. P29 asked many rhetorical questions to support his comprehension of the new task, while P32 monitored and reflected on his own thinking in the context of a complex task. P23 continuously self-questioned and reviewed his approach as he considered the general cases in his program:

> *What are the specifications? [P29] (Planning);*
>
> *Create a framework for the Date class and Test class [P32] (Planning);*
>
> *I ask myself frequently what are the general cases in each situation? [P23] (Monitoring);*
>
> *This method is difficult and I should provide for many exceptions, especially for leap years [P32] (Monitoring, Regulation).*

The type of *problem-solving strategies* used by participants in the early stages of programming also emerged as a major factor impacting on success in the final program. In most cases, participants did not state explicitly which strategy they had used; however, they recorded the steps they used during the problem-solving process and the strategy could be inferred. It is notable that different high performers used different strategies. For example, P48 used a bottom-up strategy, while P32 indicated use of a top-down strategy:

> *I create the class and declare methods [P48] (Bottom-up strategy);*
>
> *I will start with the … Date class and Test class, headings, import given methods, etc. [P32]*
>
> *(Top-down strategy).*

The theme, *Handling of errors and problems*, refers to problems and complexities that participants encountered in the development of their programs. The spontaneous comments below (which were not of a type specifically requested in the specification for the thinking-processes document) made it clear that the nature and extent of errors, as well as approaches to solving them, were crucial elements. Participants interpreted their problems and learned from their mistakes:

> *My 'if'-structure is incorrect [P48];*
>
> *I should make many exceptions for problems [P32].*

The *Additional support* theme unveiled the fact that successful students used a variety of sources and supplementary forms of support—textbooks, completed assignments; and the Internet—in their efforts to design programs, to write programming code, and to solve their problems independently.

> *I have used previous Java assignments [P44];*
>
> *I used a textbook [P32];*
>
> *I used Wikipedia and previous assignments [P29].*

An integrated theory of the themes that emerged

The development of a grounded theory is based on the conceptual clarity that occurs as it becomes increasingly possible to identify emergent patterns and theory. Figure 2 presents a final thematic pattern that was constructed from overall analysis of the participants' programs and thinking processes. This diagram can be considered as an integrated representation of the mental concepts and activities that contribute to the process of successful computer programming. Certain themes relate to the core activities involved in programming and can be viewed as the foundation of the structure, namely the use of *cognitive knowledge, skills and strategies*; *metacognitive thinking processes*; and *problem-solving expertise*. The other two themes that emerged, namely, *Handling of errors and problems*, and *Additional forms of support*, are shown as scaffolding on the side.

The propositions in Figure 2, namely, '*address*' and '*facilitate*', represent relationships that occur as scaffolding supports the programming process. These propositions indicate that the high-performing participants were able to solve their problems independently, using reflection and additional sources of information to support their efforts.

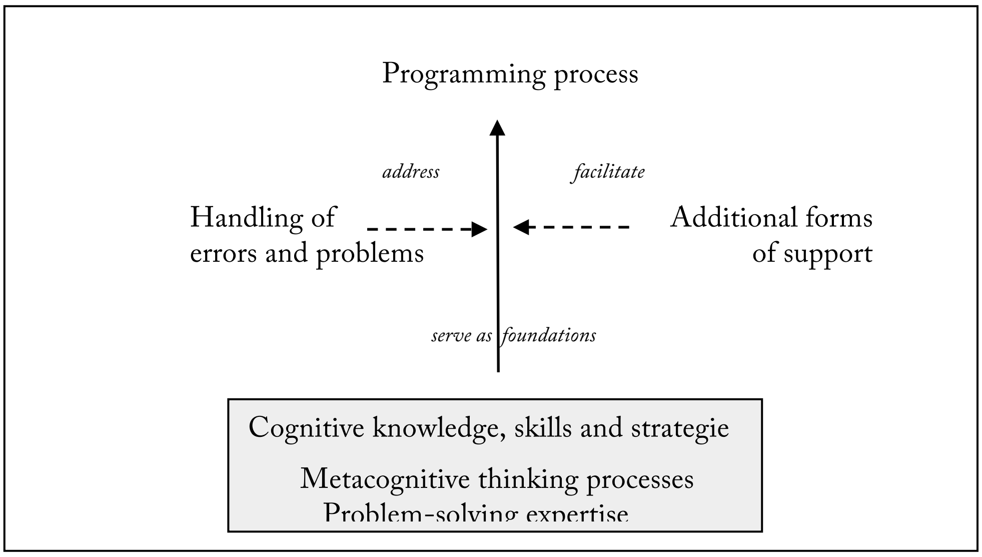
Programming process
address
facilitate
Handling of errors and problems
Additional forms of support
serve as foundations
Cognitive knowledge, skills and strategie
Metacognitive thinking processes
Problem-solving expertise

Figure 2: An integrated theory of the themes that emerged from high-performing participants' thinking processes

4.3 Examples of students' mental activities during programming

Various examples of human thought, knowledge, recall and reflection are shown in Table 1, which maps concepts from the literature of Sections 2.1 and 2.2, against mental activities of the participants. These incidences of mental activity emerged from textual data in the students' journals, which was transferred by the primary researcher to coded primary documents as described in Section 4.1. They illustrate ways in which the high performers recalled relevant factors and reasoned about the problem. The activities supported them in planning, managing and regulating their actions and enabled them to successfully design and implement their computer programs

Table 1: The mental activities of successful students during programming

Concepts from literature study: Human mind in programming	**Mental activities of participants:** **Examples from journals and programs**
Patterns or visual sketches (syntax) (Parnin, 2010).	Programming syntax: **if** (YearLeap == **true**) Days = 29; **else** Days = 28; [P32].
Semantic meaning of statements (Parnin, 2010).	[Days of] The 1st, 3rd, 5th, 7th, 8th, 10th and 12th month should be smaller than 32. The 2nd month should be smaller than 29 (except for leap years) else it [days in each month] should be smaller than 31 [P48].
Reasoning (Sternberg, 2006).	i. Test the year. Determine if it is a leap year (February). ii. Then test the months. Use arrays. iii. Now test the days.
Decision-making (Simon, 1955; Sternberg, 2006).	I think about the screen layout [P29].

Concepts from literature study: Human mind in programming	**Mental activities of participants: Examples from journals and programs**
Recall, memory and forgetting (Chen et al., 2010; Unsworth et al., 2009).	…when I realised that I should use a nested 'if' statement [to test for leap years], it was very easy [P32]. I cannot believe that I forgot how to create a [programming] class. I quickly looked in my textbook and recalled the knowledge [P48].
Attention and comprehension (Sternberg, 2006)	I read the question with attention and get an overview of what was been asked [P42].
Declarative knowledge (Ismail et al., 2010).	The input format [for dates] is : yyyymmdd [P23]. I have written two methods: one to determine leap years and the other to determine the difference between two dates [P15].
Procedural knowledge and problem solving (Ismail et al., 2010).	You require a method to copy from the date [yyyymmdd] the day, month and year…I therefore require three methods: one for *getDay*, *getMonth* en *getYear* respectively [P23].
Metacognitive knowledge (Ismail et al., 2010).	I read the question carefully and determined what was being asked? [P29]. Finally my program is working. It was a challenge. I should do it more regularly [P48].

5. Quantitative analysis

Both descriptive and inferential statistics were used.

5.1 Quantitative data analysis

The right side of Figure 1 in Section 3.1 displays the systematic investigation by which standard statistical measurements were applied during quantitative data analysis. As explained in Section 3.1, the postpositivist approach can on occasions involve both quantitative and qualitative data. We assigned scores to quantitative aspects, by rating participants' performance in the programming task in the same way we would mark (score) it for a semester mark. Similarly, the thinking processes of students, as established by the GTM analysis of Section 4.2, were analysed according to measurement criteria generated from a theoretical literature study (Havenga, 2008; Section 2.2). Table 2 shows the quantitative data, classified under four conceptual categories of criteria with 24 more specific subcriteria. Twenty-three of the criteria (left column in Table 2) were measured on a 4-point scale, where 1 indicates poor performance and 4 an excellent performance. For the problem-solving category, with a single criterion, a score out of 8 was allocated. The 24 criteria thus score a total of 100. To be classified as 'successful' in programming, participants had to obtain 3 or 4 for the 'Correctness of output' subcategory (3rd last row in Table 2), demonstrating accurate program output and appropriate test data to test the program. Using this requirement, there were 11 successful and 37 unsuccessful programmers among the 48 participants (Havenga et al., 2008). The data of the former 11 was used in this study.

5.2 Findings from the quantitative analysis

Descriptive statistics

The scores were analysed by descriptive statistics to determine the means and standard deviations of the scores of high performers for all subcategories and for the overall categories as shown in Table 2.

Table 2: Means and standard deviations of successful programmers' results

Category	Participant number											
	12	15	23	28	29	32	38	40	42	44	48	$\overline{x}$
Cognition	$\overline{x}$ =3.85 s=0.20											
Knowledge	4	4	4	4	4	4	4	4	4	4	4	4.00
Comprehension	4	4	4	4	4	4	4	4	4	4	4	4.00
Application	4	4	4	4	4	4	4	4	4	4	4	4.00
Analysis	4	3	4	4	4	4	4	4	4	4	3	3.82
Synthesis	4	3	4	4	4	4	4	3	4	4	3	3.73
Evaluation	4	3	3	4	4	4	3	3	4	4	3	3.55
Metacognition	$\overline{x}$ =3.33 s=0.54											
Planning	4	4	4	4	4	4	4	4	3	4	4	3.91
Monitoring	4	3	4	4	4	4	2	3	2	3	3	3.27
Regulation	4	2	3	3	3	4	2	2	2	3	3	2.82
Problem solving	8	8	8	8	8	8	8	8	8	8	8	8.00
OOP activities	$\overline{x}$ =3.62 s=0.29											
Program requirements analysis	4	4	4	4	4	4	4	4	4	4	4	4.00
Programming techniques	4	4	4	4	4	4	4	4	4	4	4	4.00
Programming statements	4	4	4	4	4	4	4	4	4	4	3	3.91
User-friendliness	3	3	2	3	4	4	3	2	2	4	3	3.00
Classes and objects	4	3	4	4	4	4	4	4	4	4	3	3.82
Method application	4	3	4	4	4	4	3	3	4	4	3	3.64
Access control	4	4	4	4	3	4	4	4	4	4	4	3.91
Parameter passing	4	4	4	4	4	4	4	4	4	4	4	4.00
Reasoning and logic	4	3	4	4	4	4	4	3	4	4	3	3.73
Exception handling	3	0	3	3	3	4	2	1	3	4	2	2.55
Program structure and scope	3	3	4	4	4	4	4	4	4	4	3	3.73
Solution of problem	4	3	4	4	4	4	4	3	4	4	3	3.73
Program evaluation	3	3	4	4	4	4	4	3	3	4	3	3.55
Correctness of output	3	3	3	3	4	4	3	3	3	3	3	3.18
TOTAL (%)	95	82	94	96	97	100	90	85	90	97	84	91.82
*Problem-solving strategy	BU	TD	IG	BU	IG	TD	BU	BU	BU	BU	BU	

*BU = Bottom-up; TD = top-down; IG = Integrated strategy.

The table shows that in the problem-solving category, with its single row, all the high-performing participants obtained the maximum score of 8. For cognition, metacognition and OOP activities, they obtained more than 3 on a 4-point scale across the board, except for the 'regulation' and 'exception handling' subcategories, where a few defaulted, particularly in the latter where some imperfect performances occurred.

Correlation

Furthermore, we investigated statistical correlations between cognition, metacognition and OOP constructs respectively to determine the relationship between scores assigned to the qualitative mental activities of computer science students and scores in their programming tasks. The Spearman correlations between these pairs of variables are shown in Table 3.

Table 3: Correlations between cognition, metacognition and OOP knowledge and skills of successful participants

Construct	*r*
Cognition OOP knowledge and skills	0.80*
Metacognition OOP knowledge and skills	0.55*

Practically significant (Steyn, 2002).

The cognition/OOP correlation is considerably larger than 0.5 namely 0.80. The metacognition/OOP correlation is 0.55. However, both are relevant in practice (Ellis & Steyn, 2003), indicating a highly significant relationship between the mental construct of cognition and the OOP programming construct, and a significant relationship between the mental construct of metacognition and the programming construct of OOP.

6. Discussion

6.1 Overview of the findings

Analyses of the qualitative and quantitative data revealed that high-performing computer science students employed various thinking processes to support their computer programming performance (Section 4, Tables 1 and 2). With regard to their expertise in programming, these students demonstrated high levels of knowledge with regard to the object-oriented programming approach and constructs of the programming language (*declarative knowledge*). They were skilled in using syntax and semantics, and knew how and when to apply the input, output and calculations involved in the program (*procedural knowledge*).

The application of various strategies enhanced their programming performances. The high performers effectively used cognitive, metacognitive and problem-solving skills and strategies to direct their thinking processes (Figure 2). They applied all the levels of Bloom's taxonomy and organised their thinking processes to address the programming problem (*cognitive strategies*). *Metacognition* was demonstrated by journal entries that indicated reflection on the tasks, and explained their actions and decisions on the use of specific programming statements. They employed self-management and self-evaluation strategies. Results indicate that the high-performing students regulated their thinking processes more effectively than the other students.

Furthermore, the successful programmers used various processes emanating from psychology, such as attention; comprehension; step-wise reasoning; problem-solving; decision-making; memory strategies; and recall techniques (Table 1), to express themselves in semantically correct programming code and thus to enhance their programming performance. Table 1 also gives an example of the use of visual patterns to facilitate correct syntax. The high performers were able to reflect on their errors and handle them with appropriate trouble-shooting and recovery techniques. In addition, they fostered their programming prowess by independently consulting supplementary media and forms of support.

As indicated in Table 2, these participants obtained the maximum score of 8 for problem-solving strategies and none of them used the trial-and-error approach. Most of the successful participants (7) used a bottom-up problem-solving strategy, two used top-down and two applied the integrated strategy (last row of Table 2). Successful participants applied various programming skills and gave evidence of correct output and appropriate test data. The relationships between the constructs show that cognitive and metacognitive thinking processes impacted on programming performance (Table 3). In particular, the positive correlation between cognition and OOP (r = 0.80) allows us to predict that these two variables are related. The relationship between metacognition and OOP is r = 0.55, indicating that the application of metacognitive processes and reflection can support problem solving in object-oriented programming.

6.2 Research question revisited

This study addressed the research question:

> *How can the specific thinking processes used by high-performing computer science students, support them in writing correct and well-defined programs?*

It is clear from the findings that the high-performers explicitly applied a variety of thinking processes and strategies, which helped them, produce correct and well-defined programs. A concise summary follows:

- Mental activities during the programming process were directed by declarative-, procedural- and metacognitive knowledge and skills;
- Strategy formulation and execution enabled participants to plan a high-level strategy on how to approach and solve the problem;
- The use of analysis, synthesis and evaluation skills enabled participants to decompose the problem into manageable sections; to determine solution strategies for each; and to elaborate, organise and integrate their programming statements in ways that solved the problem successfully;
- Planning, monitoring, and self-regulation strategies supported independent reviewing, self-evaluation, diagnosis and error handling—for both logic errors and programming errors;
- Supplementary forms of support were consulted; and
- A range of psychological modalities and activities enabled participants to represent, design, code and execute correct and well-defined OOP programs (modalities refer to form and to the use of visual and spatial senses, which play an important role in representing and designing programs).

This study with its *focus area* of thinking processes in the *application area* of Computer Science Education, has made a contribution in the realm of inter-disciplinary studies. We inter-related psychological dimensions — in the form of students' reflective thinking processes — with the actual outcomes of computer programming tasks. The high-performing participants orchestrated their thinking processes in a comprehensive way and demonstrated a uniformity in performance, yet with variations in techniques. The study has provided new insights regarding the efforts lying behind a complete operational computer program.

6.3 Integrated thinking processes model for learning programming

Figure 3 presents a model based on the findings of this study. Anti-clockwise, its three components are *educational activities, programming performance* and *psychological processes*—from the three disciplines respectively. The representation integrates the thinking processes, knowledge, skills, mental activities and strategies used by high-performing student programmers as they undertake a complex programming task.

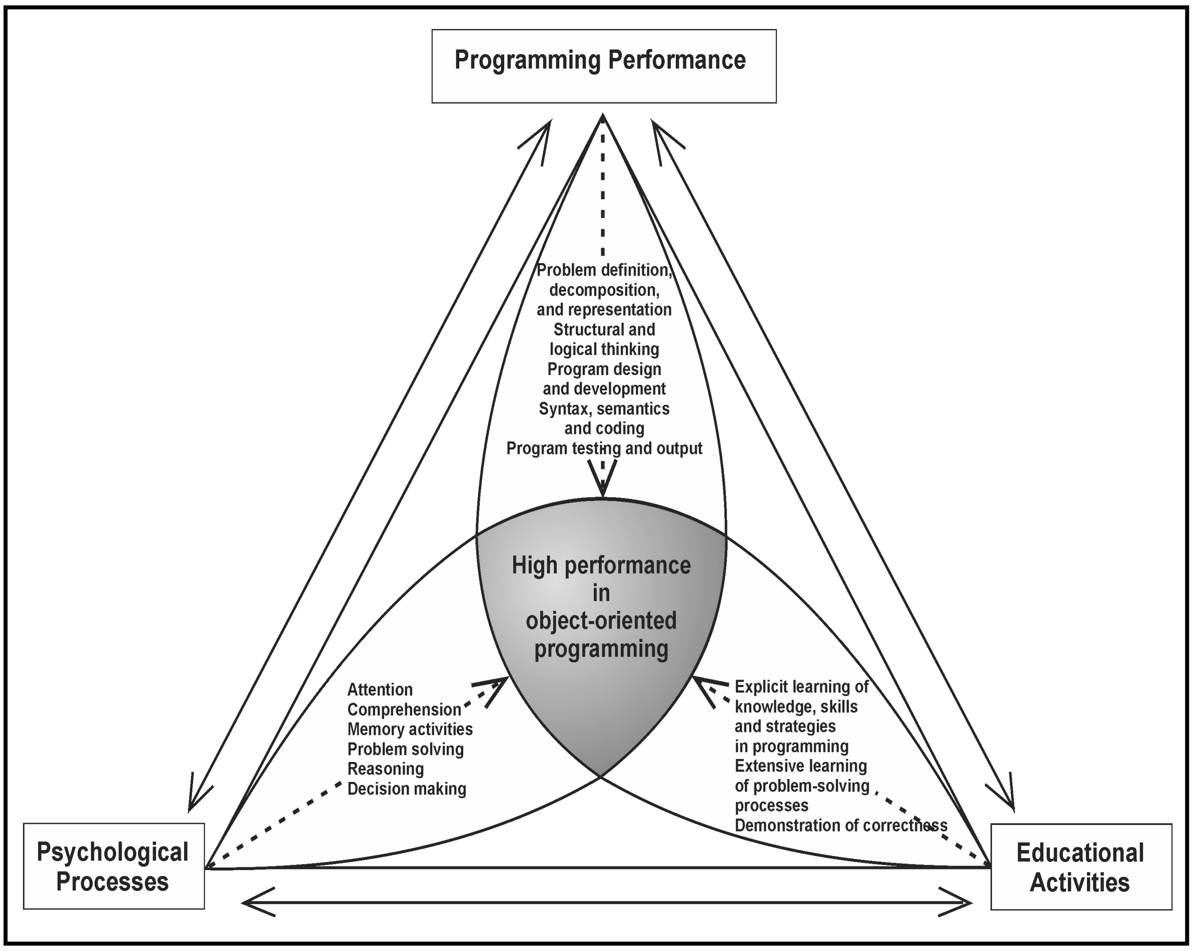
Programming Performance
Problem definition, decomposition, and representation
Structural and logical thinking
Program design and development
Syntax, semantics and coding
Program testing and output
High performance in object-oriented programming
Attention
Comprehension
Memory activities
Problem solving
Reasoning
Decision making
Explicit learning of knowledge, skills and strategies in programming
Extensive learning of problem-solving processes
Demonstration of correctness
Psychological Processes
Educational Activities

Figure 3: An integrated thinking processes model for learning programming.

The outer arrows represent interactions between the three components. The inner dotted arrows, converging on the shaded intersection, emphasise the dynamic *integration* of processes and activities that support high performance in learning object-oriented programming.

The explicit use of supportive skills and strategies, over and above content knowledge can facilitate learning to program, as shown at the right vertex of the model. Klingberg (2010) points out that extensive teaching of strategies leads to the improved performance of working memory tasks (Section 2.1 and 2.2). In a previous paper, we (Havenga et al., 2010) described how the successful use of supportive knowledge and activities can be 'actionable' in practice, and gave examples. In their interactive learning model, Tennyson and Nielsen (1998) (Section 2.2) suggest that the interaction of content knowledge with cognitive strategies can be learned. Furthermore, a sound knowledge of problem-solving processes for both well- and ill-structured programming problems should form a foundation for writing programs that execute correctly. Finally, programmers must be able to selfevaluate the accuracy and correctness of their programs to demonstrate how 'well' they have solved the problem.

The psychological processes and modalities at the left vertex should support and direct programmers' thinking as they understand, represent, design, code and test a program to solve a real-world problem. The process of computer programming involves various mental activities and qualities. For example selective attention is closely linked to working memory performance (Klingberg, 2010).

When writing programs, programmers need to think on a high level of abstraction. Many student programmers have a reasonable grasp of syntax, but struggle to holistically combine code to produce a program that executes correctly. Programming, positioned at the top vertex — with specific reference in this study to object-oriented programming — requires structured thinking on the part of the programmer to organise and sequence the segments of programming code within the program units, or so-called 'classes'. Sound detailed specification of the overall program and design activities supports the organisation of such classes. Logical and rigorous thinking is required to direct the integration of programming statements into so-called 'methods' that specify the required behaviour of the 'objects' in the program. The employment of supportive learning techniques and psychological activities assists the programmer in writing correct and well-defined computer programs.

7. Conclusion

This trans-disciplinary study investigated an integration of the domains of computer programming, education and psychology. Psychological dimensions of Computer Science students—in the form of the reflective thinking processes of high-performing programmers during the programming process—were related to the actual outcomes of their computer programs. Findings indicated that high-performing programmers used a range of mental activities and supportive strategies, applying various cognitive, reflective and psychological processes and activities. They employed sound thinking processes and orchestrated their activities in a comprehensive way, which contributed to successful completion of their programming tasks. Based on the findings, we propose a model of integrated thinking processes to support problem solving in complex tasks.

This study provides insights regarding the multi-disciplinary efforts underlying a complete computer program that executes correctly. The questions arise: Can learning strategies and higher-order thinking skills be taught and learned? Is there a place in education for the explicit teaching of supportive skills and cognitive strategies, over and above content instruction? Future research should focus on teaching practices that incorporate such strategies to facilitate learning to program.

8. References

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the Role of Self-Regulated Learning on Introductory Programming Performance. *International Computing Education Research (ICER),* 81-86.

Bloom, B.S., Krathwohl, D.R., & Masia, B.B. (1973). *Taxonomy of Educational Objectives. Book2: Affective Domain*. London: Longman Group.

Börstler, J. & Schulte, C. (2005). Teaching Object Oriented Modelling with CRC-cards and Role playing Games. *Proceedings of 8th IFIP World Conference on Computers in Education (WCCE) 2005, Cape Town, South Africa, 4-7 July 2005.*

Bryman, A. (2006). Integrating quantitative and qualitative research: how is it done? *Qualitative Research,* 6, 97-113.

Chen, K.H.M., Chuah, L.Y.M., Sim, S.K.Y., & Chee, M.W.L. (2010). Hippocampal region-specific contributions to memory performance in normal elderly. *Brain and Cognition,* 72, 400-407.

Creswell, J.W. (2008*). Educational Research. Planning, Conducting, and Evaluating Quantitative and Qualitative Research* (3rd ed.). New Jersey: Pearson Education.

Creswell, J.W. (2009*). Research Design: Qualitative, Quantitative and Mixed Methods Approaches* (3rd ed.). SAGE Publications Inc. Thousand Oaks, CA.

De Villiers, M.R. (2005). Interpretive research models for Informatics: action research, grounded theory, and the family of design- and development research. *Alternation*, 12(2), 10-52.

Ellis, S.M. & Steyn, H.S. (2003). Practical significance (effect sizes) versus or in combination with statistical significance (p-values). *Management Dynamics,* 12(4), 51-53.

Flavell, J.H. (1979). Metacognition and Cognitive Monitoring. A New Area of Cognitive Developmental Inquiry. *American Psychologist,* 34(10), 906-911.

Gibbs, G.R. (2010). *Analyzing Qualitative Data*. Los Angeles: SAGE Publications.

Glaser, B.G. & Strauss, A.L. (1967). *The Discovery of Grounded Theory. Strategies for Qualitative Research*. London: Weidenfeld and Nicolson.

Gravill, J.I., Compeau, D.R., & Marcolin, B.L. (2002). Metacognition and IT: The influence of Self-Efficacy and Self-Awareness. *Eighth Americas Conference on Information Systems. 2002: 1055-1064.*

Gunter, M.A., Estes, T.H., & Mintz, S.L. (2010). *Instruction. A Models Approach*. (5th ed.). Boston: Pearson.

Havenga, H.M. (2008). An investigation of students' knowledge, skills and strategies during problem solving in object-oriented programming (PhD Thesis: Unisa).

Havenga, M., Mentz, E., & De Villiers, R. (2008). Knowledge, skills and strategies for successful object-oriented programming: a proposed learning repertoire. *South African Computer Journal (SACJ),* 42,1-8.

Havenga, H.M., De Villiers, M.R., & Mentz, E. (2010). Enhancing the thinking processes of computer programming students who are average performers. In *Proceedings of the 1st International Conference on Mathematics, Science and Technology Education (ISTE) 2010: 64-78.* Mopani Camp, Kruger National Park. October 2010.

Ismail, M.N., Ngah, N.A., & Umar, I.N. (2010). Instructional Strategy in the Teaching of Computer Programming: A need assessment analysis. *The Turkish Online Journal of Educational Technology,* 9(2), 125-131.

Klein, H.K. & Myers, M.D. (1999). A set of principles for conducting and evaluating interpretive field studies in Information Systems. *MIS Quarterly*, 23(1), 67-94.

Klingberg, T. (2010). Training and plasticity of working memory. *Trends in Cognitive Sciences,* 14, 317-324.

Matavire, R. & Brown, I. (2008). Investigating the Use of "Grounded Theory" in Information Systems Research. In *Riding the Wave of Technology, Proceedings of South African Institute for Computer Scientists and Information Technologists (SAICSIT) 2008: 139-147. ACM International Conference Proceedings Series. George, October 2008.*

Muhr, T. (2004). User's Manual for ATLAS.ti 5.0. (2nd ed.). Retrieved 3 September, 2010, from http://www.atlasti.com/downloads/atlman.pdf

Onwuegbuzie, A.J., Johnson, R.B., & Collins, K.M.T. (2009). Call for mixed analysis: A philosophical framework for combining qualitative and quantitative approaches. *International Journal of Multiple Research Approaches*, 3, 114-139.

Parnin, C. (2010). A Cognitive Neuroscience Perspective on Memory for Programming Tasks. Retrieved 27 October, 2010, from http://www.cc.gatech.edu/~vector/papers/memory.pdf

Sebesta, R.W. (2008). *Concepts of Programming Languages.* (8th ed.). Boston: Pearson Addison Wesley.

Simon, H.A. (1955). A Behavioral Model of Rational Choice. *The Quaterly Journal of Economics,* 69(1), 99-118.

Sternberg, R.J. (2006). *Cognitive Psychology* (4th ed.). United Kingdom: Thomson Wadsworth.

Steyn, H.S. (2002). Practically significant relationships between two variables. *SA Journal of Industrial Psychology,* 28(3), 10-15.

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

Strauss, A. & Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded theory* (2nd ed.). Sage Publications, Thousand Oaks: CA.

Tennyson, R.D. & Nielsen, M. (1998). Complexity Theory: Inclusion of the Affective Domain in an Interactive Learning Model for Instructional Design. *Educational Technology*, 38(6), 7-12.

Unsworth, N., Redick, T.S., Heitz, R.P., Broadway, J.M., & Engle, R.W. (2009). Complex working memory span tasks in higher-order cognition: A latent-variable analysis of the relationship between processing and storage. *Memory,* 17(6), 635-654.

Webb, N.L. (2002). Alignment study in language arts, mathematics, science, and social studies of state standards and assessments for four states. Washington, DC: Council of Chief State School Officers.