Knowledge, skills and strategies for successful object-oriented programming: a proposed learning repertoire

Marietjie Havenga¹, Elsa Mentz², Ruth de Villiers³

Computer Science Education, North-West University, Private bag x601, Potchefstroom
School of Computing, University of South Africa, P O Box 392, Unisa, 0003

ABSTRACT

Third year Computer Science students were studied in order to determine which knowledge, skills and strategies they used during an object-oriented programming task. Quantitative and qualitative methods were used to analyse their computer programs and associated thinking processes. Successful programmers applied significantly more cognitive, metacognitive and problem-solving knowledge, skills and strategies, also using a greater variety, than the unsuccessful ones. Based on the approaches of the successful programmers, we propose a learning repertoire of integrated knowledge, skills and strategies, which can serve as a framework to support novices learning object-oriented programming (OOP).

CATEGORIES AND SUBJECT DESCRIPTORS

D.3.3 [Programming Languages]: Language Constructs and Features – abstract data types, classes and objects.

GENERAL TERMS

Languages, Design, Performance, Reliability.

KEYWORDS

Object-oriented programming, successful, unsuccessful programmers, thinking processes.

1. INTRODUCTION

Learning and conducting object-oriented programming (OOP) is multidimensional and complex [15]. OOP requires the use of specific knowledge, skills and strategies to solve problems and write the associated programs. Successful and unsuccessful programmers differ in the way they approach and solve programming problems. An unsuccessful programmer is a person who did not achieve the stated outcomes, while a successful programmer is one who did achieve them and who dealt efficiently with problems [15]. Successful programmers possess a well-organised, carefully-learned knowledge structure [1]; they use self-regulatory processes and monitor their problem-solving activities [14] and they can solve a problem quickly, although they often appear to spend more time in problem representation [25].

These are some examples of cognitive, metacognitive and problem-solving activities that are required in programming. However, these are not merely personal or isolated learning techniques, but rather distinct activities that should explicitly be integrated to address a programming problem and solve it successfully. This paper considers the following research questions: What are the differences between the ways that successful and unsuccessful programmers apply their knowledge, skills and strategies in an object-oriented programming task? How can novices be supported in learning OOP?

The objective of the first question was an attempt to identify cognitive, metacognitive and problem-solving knowledge, skills and strategies used by successful and unsuccessful programmers in OOP. To answer the second, we attempted to integrate the approaches of successful programmers into a learning repertoire that can serve as a framework for novices learning OOP.

2. LITERATURE SURVEY

Computer programming involves a rich environment in which specific programming words, statements and constructs come together to be integrated in a tightly defined way to solve a problem efficiently. This requires high-level knowledge, skills and strategies. In general, the knowledge relates to information and skills acquired through experience or education. A skill refers to the ability to do a particular task, while a strategy is a designed plan to achieve a purpose and to solve a problem [6]. It is often assumed that students implicitly and independently master the required high-level knowledge, skills and strategies, and that teaching should focus on programming content and coding structures only. However, to be successful in the complex domain of OOP, explicit learning...
of both facets is required. This survey briefly overviews some aspects and techniques that can support successful programming.

2.1 Cognition

The concept of cognition refers to the mental processes used in the acquisition, storage, transformation and application of knowledge [25]. In this regard Bloom’s taxonomy [3] defines six types of learning, hierarchically ordered according to the level within the cognitive domain: knowledge; comprehension; application; analysis; synthesis; and evaluation. The way in which these concepts are used (or not used) can define the differences between successful and unsuccessful programmers [28], where the six associated skills are, respectively: knowledge of the programming language; interpretation of the programming problem; application of prior knowledge in a new program; analysis of the problem; design of a new program; and evaluation of the solution. Since programming is ‘extremely cumulative’, novices must progress through each of Bloom’s six levels to become truly successful [4] [28].

Recall of information can be improved by cognitive strategies [22] such as rehearsal, elaboration and organisation [2]. Rehearsal strategies, for example: focussing attention, structured recall, and distributed practice over a period of time; can support recollection and help to pinpoint important information within a context. In the programming context, programmers who repetitively sequence activities in a particular way ‘preserve the effect’, using less working capacity [17]. Elaboration helps students to integrate new information with prior knowledge by, for example: generative note taking, asking questions, summarising, and creating analogies. The organisation strategy includes extraction of the main idea from text as well as integration of concepts [2] with the goal of achieving a holistic problem solution.

2.2 Metacognition

Metacognitive knowledge is explicit knowledge of one’s own cognitive strengths and weaknesses, beliefs and conditions that affect memory performance [16][21]. Self-knowledge, task-knowledge and strategy knowledge are required in the metacognitive domain [11]. Metacognitive strategies include planning, monitoring and regulation. In programming, planning entails analysis of the problem and the identification of possible classes and methods to solve it, while monitoring guides the process of finding a solution by means of self-testing [2]. Regulation involves the continuous modification of one’s cognitive activities to determine whether the problem is being solved successfully. Bergin et al. [2] discuss self-regulated learning with regard to the performance of students in their third level of introductory OOP. They found that students with high levels of intrinsic motivation perform better and use more metacognitive-management strategies than lower performing students.

2.3 Problem solving

Different kinds of problems are solved in different ways and require different approaches. Students should understand how problems vary according to their structuredness, complexity, dynamicity and domain-specificity [20]. In this regard, programming experience and exposure play roles and Sternberg [25] suggests that experts develop sophisticated internal representations of certain kinds of problems, based on their structural similarities. Standard problem-solving strategies are: bottom-up, top-down, integrated, as-needed and trial-and-error [7] [9] [29]. Research shows that expert object-oriented programmers tend to use top-down strategies during the early phases of programming to understand systems holistically. In contrast, the same experts may use a bottom-up strategy when programming in an unfamiliar context or during program maintenance where individual parts are combined to form larger components [7].

2.4 Object-oriented programming

OOP is based on the object-oriented approach, where objects are models of real-world entities that have the responsibility of carrying out specific tasks to solve the problem [12]. OOP involves various knowledge and skills relating to data types, control structures, instantiation of objects, methods, GUI tools, exception handling, database connectivity [19], input/output validation, performance correctness [24], debugging and the development of test data. Due to the complexity of OOP, students have difficulty in applying the required activities successfully [15]. Explicit teaching and learning of high-level knowledge, skills and strategies may therefore be a requirement to support success in OOP.

3. RESEARCH DESIGN

The underlying research ethos of this study is constructivist problem solving, which refers to the students’ active construction of computer programs and application of programming constructs such as classes and objects. It also relates to the researcher’s construction of a body of knowledge regarding the students’ programming constructs, as she interprets and reflects on those programming experiences. This implies a continuous process of interpretation and reflection.

In a mixed methodology, both quantitative and qualitative research methods were used to analyse participants’ computer programs and the associated written thinking processes. Quantitative methods include statistical calculations such as descriptive statistics, practical significance and correlation. As a qualitative research practice, grounded theory was applied to guide the systematic collection of data and to generate a model inductively from the ongoing data collection and analysis to explain the specific phenomenon [8][13].

3.1 Data collection

The research was conducted over a period of two years. The participants (n = 48) came from two groups: the first group, namely 2005, consisted of 11 BEd and 17 BSc 3rd year students, and the second group, namely 2006, comprised three BEd and 17 BSc 3rd year students. Students from both groups took Computer Science as a major subject. Each participant had to create an object-oriented program relating to leap years. It was an open-ended question and participants had to decide personally which calculations were necessary in the program. However, some requirements were included to direct the programming process. At the very least, the students should write a Date class program to calculate which years are leap years and the difference between any two dates in the range 1 January 1800 to a later date. A Test class program was also required to determine whether the output of the Date class was correct. The programs could be done in either Delphi or Java. During the major process of programming the Date class task, participants were required to record their thinking and problem-solving processes in writing.

Data collection included both the computer programs and the recorded thinking processes. Triangulation was applied by investigating data from these two sources, i.e. the coded programs and the associated thinking processes written by participants as they considered the problem and coded their
solutions. Finally, coherence between the different data sources was investigated to identify patterns of meaning and to describe the emerging theory that leads to the learning repertoire.

3.2 Data analysis

Two approaches were followed. In the first approach, each program itself and the recorded thinking processes were evaluated, using as an instrument, a set of measurement criteria that had emerged from the literature review. The 24 criteria (or subcategories) shown in Table 1 originate from four major categories: cognitive knowledge and skills; metacognitive strategies; problem-solving strategies; and OOP knowledge and skills. Measurement of 23 of the criteria was scored on a 4-point scale where 1 indicates poor performance and 4 an excellent performance. For the problem-solving category with its single criterion, participants could use more than one strategy, so a maximum of 8 was allocated instead of 4. Participants who used the trial-and-error strategy received zero, since it was not considered an acceptable problem-solving strategy. The 24 criteria thus score a total of 100. As the indicator of ‘successful’ programming, participants had to obtain 3 or 4 for the ‘Correctness of output’ subcategory (last criterion in Table 1), relating to evidence of correct program output and the test data used. Based on this approach, there were 11 successful and 37 unsuccessful programmers.

The scores were analysed by descriptive statistics to determine the means and standard deviations of successful and unsuccessful participants for all criteria and for the overall categories. Practical significant differences (effect size) between successful and unsuccessful participants were determined for all criteria, as shown in Table 2. Guidelines for the interpretation of effect size are as follows: $d = 0.2$ small effect; $d = 0.5$ medium effect; $d = 0.8$ large effect [5]. Values $\geq 0.8$ mean that the effect size of constructs is regarded as practically significant [10]. However, Thompson [27] warns that researchers should avoid using these guidelines in an overly rigid way. In order to determine correlations between the cognitive, metacognitive and OOP constructs, the Spearman ranked correlation coefficient was used, as shown in Table 3. The correlation is interpreted as follows: $r = 0.1$ small effect; $r = 0.3$ medium effect; and $r = 0.5$ large effect [5]. Data with an $r$ value $\geq 0.5$ is considered as practically significant [10].

The second analysis approach investigated the thinking processes of participants, using the qualitative analytical software package, Atlas.ti. The purpose was to identify various themes that emerged from the recorded thinking processes. The researcher allocated codes to particular segments in the typed textual data until sufficient similar patterns were identified, indicating that saturation had occurred. After the codes were grouped and categorised, various themes were identified.

<table>
<thead>
<tr>
<th>Category</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cognitive knowledge and skills</td>
<td>Knowledge (4) Evidence of knowledge of the programming language</td>
</tr>
<tr>
<td></td>
<td>Comprehension (4) Interpretation of the problem</td>
</tr>
<tr>
<td></td>
<td>Application (4) Application of prior knowledge in a new program</td>
</tr>
<tr>
<td></td>
<td>Analysis (4) Analysis of the problem – breaking it down into steps</td>
</tr>
<tr>
<td>Metacognitive strategies</td>
<td>Synthesis (4) Designing a new program</td>
</tr>
<tr>
<td></td>
<td>Evaluation (4) Evaluation of the solution</td>
</tr>
<tr>
<td>Planning</td>
<td>Evidence of planning during programming</td>
</tr>
</tbody>
</table>

Table 2. Means, standard deviations and practical significances for unsuccessful and successful participants

<table>
<thead>
<tr>
<th>Category</th>
<th>Unsuccessful participants (37)</th>
<th>Successful participants (11)</th>
<th>Practical significance (effect size)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$ s</td>
<td>$\bar{x}$ s</td>
<td></td>
</tr>
<tr>
<td>Cognition</td>
<td>3.05 0.71</td>
<td>3.85 0.20</td>
<td>1.13*</td>
</tr>
<tr>
<td>Knowledge</td>
<td>3.65 0.68</td>
<td>4.00 0.00</td>
<td>0.51</td>
</tr>
<tr>
<td>Comprehension</td>
<td>3.54 0.65</td>
<td>4.00 0.00</td>
<td>0.71</td>
</tr>
<tr>
<td>Application</td>
<td>3.32 0.78</td>
<td>4.00 0.00</td>
<td>0.87*</td>
</tr>
</tbody>
</table>

---

Table 1. Measurement criteria and associated categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem-solving strategies (8)</td>
<td>Application of problem-solving strategies: bottoms up, top-down, integrated, as-needed</td>
</tr>
<tr>
<td>OOP knowledge and skills</td>
<td>Analysis of the program requirements</td>
</tr>
<tr>
<td></td>
<td>*Programming techniques used: indentation, readability, variable names and declaration</td>
</tr>
<tr>
<td></td>
<td>*Application of the correct use of programming statements</td>
</tr>
<tr>
<td></td>
<td>Application of methods such as constructors, mutators and accessors</td>
</tr>
<tr>
<td></td>
<td>*Decision on the accessibility: public, private</td>
</tr>
<tr>
<td></td>
<td>*Application of parameter passing: number, order, type of variables</td>
</tr>
<tr>
<td></td>
<td>Application of reasoning skills in OOP</td>
</tr>
<tr>
<td></td>
<td>*Application of exception handling</td>
</tr>
<tr>
<td></td>
<td>Application of program structure and scope</td>
</tr>
<tr>
<td></td>
<td>Actual solution to the problem</td>
</tr>
<tr>
<td>Correctness of output (4)</td>
<td>Evaluation of the Date class and Test class</td>
</tr>
<tr>
<td>TOTAL (%)</td>
<td>Evidence of correct program output and test data used</td>
</tr>
</tbody>
</table>

*Criteria selected specifically to reflect on general characteristics of programming (Sebesta, 2004:8).

3.3 Quantitative findings re participants’ programs and thinking processes

Table 2 summarises the measurement criteria for each category and its subcategories, giving the: mean values, standard deviations and effect size for successful and unsuccessful participants, respectively. The means for cognition, metacognition and OOP are higher for successful participants than for the unsuccessful. Practical significant differences with a large effect size were found between successful and unsuccessful participants within all subcategories except for knowledge, comprehension, classes and objects, access control and parameter passing, where practical significant differences of a medium effect size occurred.
Table 2. Means, standard deviations and practical significances for unsuccessful and successful participants continues

<table>
<thead>
<tr>
<th>Category</th>
<th>Unsuccessful participants (37)</th>
<th>Successful participants (11)</th>
<th>Practical significance (effect size)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$</td>
<td>s</td>
<td>$\bar{x}$</td>
</tr>
<tr>
<td>Synthesis</td>
<td>2.62</td>
<td>0.92</td>
<td>3.73</td>
</tr>
<tr>
<td>Evaluation</td>
<td>2.05</td>
<td>0.97</td>
<td>3.55</td>
</tr>
<tr>
<td>Metacognition</td>
<td>2.36</td>
<td>0.88</td>
<td>3.33</td>
</tr>
<tr>
<td>Planning</td>
<td>3.24</td>
<td>0.83</td>
<td>3.91</td>
</tr>
<tr>
<td>Monitoring</td>
<td>2.19</td>
<td>1.13</td>
<td>3.27</td>
</tr>
<tr>
<td>Regulation</td>
<td>1.65</td>
<td>1.06</td>
<td>2.82</td>
</tr>
<tr>
<td>OOP constructs</td>
<td>2.44</td>
<td>0.77</td>
<td>3.62</td>
</tr>
<tr>
<td>Program requirements analysis</td>
<td>3.24</td>
<td>0.86</td>
<td>4.00</td>
</tr>
<tr>
<td>Programming techniques</td>
<td>3.11</td>
<td>0.97</td>
<td>4.00</td>
</tr>
<tr>
<td>Programming statements</td>
<td>3.08</td>
<td>1.04</td>
<td>3.91</td>
</tr>
<tr>
<td>User-friendliness</td>
<td>1.62</td>
<td>1.30</td>
<td>3.00</td>
</tr>
<tr>
<td>Classes and objects</td>
<td>2.97</td>
<td>1.12</td>
<td>3.82</td>
</tr>
<tr>
<td>Method application</td>
<td>2.70</td>
<td>0.94</td>
<td>3.64</td>
</tr>
<tr>
<td>Access control</td>
<td>3.19</td>
<td>1.08</td>
<td>3.91</td>
</tr>
<tr>
<td>Parameter passing</td>
<td>3.24</td>
<td>1.12</td>
<td>4.00</td>
</tr>
<tr>
<td>Reasoning</td>
<td>2.89</td>
<td>0.81</td>
<td>3.73</td>
</tr>
<tr>
<td>Exception handling</td>
<td>0.46</td>
<td>0.80</td>
<td>2.55</td>
</tr>
<tr>
<td>Program structure and scope</td>
<td>2.86</td>
<td>0.88</td>
<td>3.73</td>
</tr>
<tr>
<td>Successful programming</td>
<td>2.41</td>
<td>1.09</td>
<td>3.73</td>
</tr>
<tr>
<td>Program evaluation</td>
<td>2.00</td>
<td>1.08</td>
<td>3.55</td>
</tr>
<tr>
<td>Correctness of output</td>
<td>0.35</td>
<td>0.72</td>
<td>3.18</td>
</tr>
</tbody>
</table>

$d = 0.8$, large effect size; $d = 0.5$, medium effect size (Ellis and Steyn, 2003:51)

Table 3. Correlations between cognition, metacognition and OOP constructs

<table>
<thead>
<tr>
<th>Construct</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cognition</td>
<td>0.63**</td>
</tr>
<tr>
<td>Metacognition</td>
<td>0.89**</td>
</tr>
<tr>
<td>OOP construct</td>
<td>0.73**</td>
</tr>
</tbody>
</table>

** Practically significant (Steyn, 2002).

3.4 Analysis of the thinking processes with Atlas.ti

Five main themes emerged in an inductive grounded-theory approach from the analysis of the participants’ thinking processes in association with their programming of the Date class, namely: cognitive knowledge, skills and strategies; metacognitive knowledge, skills and strategies; problem-solving knowledge, skills and strategies; errors and problems in programming; and additional support in programming.

3.4.1 Theme 1: Cognitive knowledge, skills and strategies

The unsuccessful participants did not refer to explicit evaluation skills as in Bloom’s taxonomy nor to cognitive strategies. Responses indicating that they used some of the skills in Bloom’s taxonomy are: I find out when it is a leap year [P31]*; I first determine the requirements [P20]; Which variables do I need? [P30]. Firstly, I thought about the class structure [P10]; Which methods should be in the class? [P21]; I need a method to convert the number of days [P36]. *P31 refers to Participant 31, etc.

Successful participants applied the full set of skills from Bloom’s taxonomy, some examples being: A programmer should understand basic principles [P15]; I received the date as a string and separated it into days, months and years [P40]. During synthesis and evaluation, participants integrated various methods in the class: I also need a method to test for valid dates [P23]. Participant 40 referred to evaluation skills when he indicated that his program was working 100%. Participant 23 applied the elaboration strategy in the following statement: When designing the class, I ask myself about the general and special cases in each situation.

3.4.2 Theme 2: Metacognitive knowledge, skills and strategies

Unsuccessful participants reflected and acknowledged their programming weaknesses. Two examples are: I have the correct idea but cannot apply it [P5]; I do not have a plan … [P34]. Some useful responses of unsuccessful participants about metacognitive strategies are: I re-read the question with attention [P30]; I could send the date to the constructor [P33]; I forgot to insert close brackets [P41]; I have determined the difference in days but was incorrect with one day [P39].

Successful participants applied a spectrum of metacognitive activities: I read the question carefully and determined what was being asked? What are the specifications? [P29]. Participant 32 used planning, monitoring and regulation strategies: Many questions were asked to determine the purpose, parameters, input, output, and problems of the programming task (planning). He also reflected on the programming task: Problems? Many! The method was difficult … and I should include many exceptions for leap years. The biggest problem was the difference between days. I have a few
ArrayOutOfBoundsException exceptions. This was solved with diagrams
(monitoring and regulation).

3.4.3 Theme 3: Problem-solving knowledge, skills
and strategies

Unsuccessful participants found it difficult to follow specific
steps during problem solving: I do not know if it is correct. I
have typed all the things that I thought should be in the program
[P31]. I … will try to code by means of trial-and-error [P34].
Participant 6 used the bottom-up strategy to solve the problem:
I will complete the code for a specific component before
continuing with the next component. Successful participants
described their systematic problem-solving steps in more detail.
For example: I determine the input, design the interface and
basic components, process and then test the input [P44].
Participant 32 used the top-down strategy when he indicated:
I will start with the framework for the Date and Test class,
headings, import given methods, etc.

3.4.4 Theme 4: Errors and problems in
programming

This theme highlights examples of errors and problems, some of
which also relate to a lack of metacognitive strategies.
Unsuccessful participants pointed out: I wonder why I typed
some of this code, because I will not use it [P39]; …exception
handling is complicated [P33]. Some participants could not
apply exception handling or interpret errors [P31, P33]; others
used incorrect syntax [P39] and could not compile the program.
Successful participants were able to diagnose and correct
t heir errors. Two examples from P32: I had problems
determining a specific date format [P32]; …the Difference() method was difficult and I should provide for many
exceptions… [P32].

3.4.5 Theme 5: Additional support in programming

Both unsuccessful and successful participants referred to
supplementary means of support during the programming
process: I used...previous code [P48]; textbooks [P30];
...previous...assignments [P44]; and Wikipedia.com for the
requirements of leap years [P29].

4. RESEARCH QUESTIONS REVISITED

This section answers the first research question: What are the
differences between the ways that successful and unsuccessful
programmers apply their knowledge, skills and strategies in an
object-oriented programming task? The answer relates to the
three major themes that emerged from the grounded theory
analysis: cognitive-, metacognitive- and problem-solving
knowledge, skills and strategies that unsuccessful and successful participants apply/do not apply in the process of a
programming task.

4.1 Emerged themes

We discuss three major themes that emerged from the grounded
theory approach.

4.1.1 Cognitive knowledge, skills and strategies

Unsuccessful participants battled to decompose the problem
scenario and to relate subparts to the overall structure. With
regard to actual programming, they could not readily apply
higher-order thinking skills. Although they used knowledge and
comprehension skills, their programs indicate that they
debugged and evaluated the code without using detailed
application and analysis skills. As a consequence, they had
problems in interpreting their errors, they could not complete
the program, and many did not obtain output.

For the higher-order thinking skills (analysis, synthesis and
evaluation) required for programming, the successful
participants received a mean value of more than 3.5 on a 4-
point scale. Their ability to apply all the levels of Bloom’s
taxonomy in a task was clear and they achieved a high level of
accuracy in solving the problem. It is notable that they spent
more time on the analysis phase and differentiated how parts are
inter-related in the complete program. Their performances
illustrate that programmers should understand the problem
precisely, interpret and evaluate their programming solutions.

Only one successful participant explicitly mentioned a
cognitive strategy that was used during programming. Possible
reasons could be that participants did not verbalise knowledge
about these strategies, they did not use cognitive strategies, or
they did not know how to apply such strategies in
programming. In this regard, Bergin et al. [2] show that
cognitive strategies are not as useful in the learning of
introductory OOP as they are in other domains.

4.1.2 Metacognitive knowledge, skills and
strategies

Unsuccessful participants found it difficult to apply
metacognitive activities during programming; they encountered
problems in monitoring and regulating their cognitive
resources. Very few of them applied any form of regulatory
strategy. They could not easily reflect on the task and their own
understanding of it, and found it difficult to manage their
thinking and reasoning.

By using detailed planning strategies, successful participants
were able to complete their tasks and produced high quality
solutions. Most participants monitored their progress and
effectively managed their cognitive resources in the process of
finding a solution (Table 2). The regulation strategy of
successful participants was slightly lower than 3 (X = 2.82),
which implies that they could improve further on regulatory
strategies during programming. These findings correspond with
Hertzog and Robinson [18], who suggest that monitoring plays
a vital role in cognitive performance of complex problem
solving and guides the process of finding a solution.

4.1.3 Problem-solving knowledge, skills and
strategies

Unsuccessful participants did not obtain the required program
output. Some encountered problems in systematically applying
problem-solving strategies. Instead, they spent time iterating
through their programming code to address errors, without
understanding which sections were incorrect and how to rectify
them. Such participants were much less accurate in their efforts
to reach an appropriate solution. Although most of the
unsuccessful participants used a bottom-up strategy (27), some
wrote that they worked without using any specific problem-
solving strategies (2). Two used trial-and-error, three used a
top-down strategy, and three used the integrated strategy.

Successful participants had considerable domain knowledge
and highly efficient problem-solving skills, which they were
able to apply successfully in the task. Seven of them used the
bottom-up strategy, two the top-down, and two the integrated
strategy during program comprehension. None of the
successful participants used the trial-and-error strategy. This
appears to indicate that it is not a successful approach in OOP,
whereas all the other problem-solving strategies were used
effectively. The second research question is: How can novices
be supported in learning OOP? It is answered by presenting a proposed learning repertoire.

4.2 Proposed learning repertoire

The constructivist problem-solving approach supports active involvement of students in constructing computer programs and applying constructs such as classes and objects. This paradigm also acknowledges the researcher’s part in the construction of knowledge about the programming constructs of students, where action, interpretation and reflection are vital.

Educators need to play supportive roles that facilitate the acquisition of appropriate activities as students learn to apply the sum of their knowledge, skills and strategies in programming. OOP is a dynamic and constructive process involving various actions and dimensions. Since its complexity can be overwhelming, we propose a learning repertoire in Figure 1 to serve as an integrated framework to support novices in learning OOP. The content of the repertoire is drawn from the empirical research, which highlights ways in which successful participants solved the programming problem. Subsets of the repertoire can be selected and used for a particular context or task.

Various dimensions are integrated in the repertoire, which explicitly distinguishes between knowledge and skills on the one hand, and strategies on the other. Knowledge and skills form the core. Cognitive knowledge and skills on all levels of Bloom’s taxonomy are required for the understanding, designing, coding and testing of a programming problem. Specific emphasis is placed on the high-order thinking skills such as analysis, synthesis and evaluation. Setting of goals, a high level of motivation, and knowledge about specific tasks are required in the metacognitive domain. In addition, adequate programming knowledge and skills are essential to the ability to complete a new program successfully.

Dynamic interaction, indicated by the arrows in Figure 1, occurs between the core sections of cognitive, metacognitive and problem-solving activities. As an example, successful object-oriented programming requires the ‘application’ of skills from Bloom’s taxonomy, particularly synthesis and evaluation to determine whether a program is correct and to rectify it if not. The dimensions in Figure 1 are supported by strategies lying outside the core. Students can use these strategies to enhance the acquisition of knowledge and skills, and can apply them during the processes of Construction, Reflection, Selection and Application in OOP. The three dashed arrows on the left, the right and below the core indicate the dynamic and continuous use of cognitive, metacognitive and problem-solving strategies in the first three processes, while the bold arrow above the core relates to the application of these activities in designing new programs and maintaining existing ones.

• Construction

The use of cognitive strategies can enhance acquisition of the knowledge and skills in Bloom’s taxonomy. Rehearsal supports the learning of facts about OOP (knowledge) and the grasping of programming content (comprehension). Elaboration can facilitate the use of previously-learned material in new situations (application) and the decomposition of a problem into subproblems (analysis). The organisation-and-integration strategy can support programmers in combining objects, methods and attributes in a class (synthesis) to program and test the correct solution (evaluation). Object-oriented programmers should be actively involved in their tasks, using prior knowledge and applying a repertoire of knowledge and skills to help them recall information and organise it in memory during the process of constructing a program.

• Reflection

Students should reflect on their cognitive processes during OOP by conducting deliberate planning, monitoring and regulation. They should question themselves, discover misconceptions, identify errors and continuously modify their programs in order to succeed. Such reflection places them in control of the programming task as they explicitly query the correctness of their code and reflect on their prior thinking to identify errors and correct flaws. Appropriate responses to feedback and the continuous improvement of code help to optimise the solution and to achieve the required outcomes.

• Selection

The ability to make discerning selections, helps students to choose a suitable problem-solving strategy for a given problem. They may select and apply one or more problem-solving strategies during program comprehension to help them to reach specific goals. For example, effective use of a top-down strategy demonstrates that a student has holistically conceptualised the entire program involving multiple classes, instances, and methods.

• Application

Finally and, in consolidation, the construction, reflection, and selection of knowledge, skills and strategies must be applied in OOP tasks to develop new programs and maintain existing ones. It is not the intention that every strategy should be applied in every situation. The various forms of knowledge, skills and strategies are relevant to different contexts. Learning to program is an active process of knowledge construction, reflection, and selection of appropriate activities to ensure successful programming.
Learning OOP requires a balanced approach of all the different activities involved. This implies, for example, that the application of Bloom’s skills without explicit reflection; or the application of strategies without any analysis, synthesis and evaluation skills will not support successful completion of a new program. In such cases, students must explicitly query the correctness of their own code and reflect on their prior thinking to identify the errors and to correct flaws.

5. CONCLUSION

To be successful in OOP, programmers require explicit learning both of programming content and higher-order mental activities. The findings of this research, which distinguishes between successful and unsuccessful programmers, indicate the need for a framework to support novice programmers. This should address programming subject matter as well as cognitive, metacognitive and problem-solving knowledge, skills and strategies. Fostering awareness and application of the latter among learners sets a particular challenge to educators (lecturers) to identify creative and effective means of doing so.

We propose a learning repertoire that includes knowledge, skills and strategies used by successful programmers. In order to apply this, various activities should occur during programming to meaningfully construct, explicitly reflect on, and critically select appropriate knowledge, skills and strategies to understand, design, code and test high quality programs. This involves the integration of specific cognitive, metacognitive and problem-solving techniques in a balanced manner. Although this framework focuses mainly on OOP, we believe that it can also be applied to support students in other programming paradigms, such as procedural programming. However, due to the particular complexities of OOP, the framework focuses specifically on a holistic view where various different decisions are required in programming one or more classes.

Future work will concentrate on the role of a lecturer or facilitator in the explicit teaching of the required knowledge, skills and strategies, supporting them in creating an educational environment in which the learning repertoire can be effectively applied. The development of assessment criteria to test the effective application of the activities of the learning repertoire in an OOP task should further support the students.

GLOSSARY

Novice: a person who is inexperienced and new in a particular field
Expert: a knowledgeable person with superior skills in a particular field

REFERENCES


