



Decoding the underlying cognitive processes and related support strategies utilised by expert instructors during source code comprehension

Pakiso J. Khomokhoana , Liezel Nel 

Department of Computer Science and Informatics, University of the Free State, Bloemfontein, South Africa

ABSTRACT

Many novice programmers fail to comprehend source code and its related concepts in the same way that their instructors do. As emphasised in the Decoding the Disciplines (DtDs) framework, each discipline (including Computer Science) has its own unique set of mental operations. However, instructors often take certain important mental operations for granted and do not explain these ‘hidden’ steps explicitly when modelling problem solutions. A clear understanding of the underlying cognitive processes and related support strategies employed by experts during source code comprehension (SCC) could ultimately be utilised to help novice programmers to better execute the cognitive processes necessary to efficiently comprehend source code. Positioned within Step 2 of the DtDs framework, this study employed decoding interviews and observations, followed by narrative data analysis, to identify the underlying cognitive processes and related support (though often ‘hidden’) strategies utilised by a select group of experienced programming instructors during an SCC task. The insights gained were then used to formulate a set of important cognitive-related support strategies for efficient SCC. Programming instructors are encouraged to continuously emphasise strategies like these when modelling their expert ways of thinking regarding efficient SCC more explicitly to their novice students.

Keywords: Source code comprehension, cognitive processes, cognitive-related support strategies, decoding the disciplines, Computer Science Education, novice programmers

Categories: • Social and professional topics ~ Computer science education

Email:

Pakiso J. Khomokhoana khomokhoanap@ufs.ac.za,
Liezel Nel nell@ufs.ac.za (CORRESPONDING)

Article history:

Received: 10 March 2020
Accepted: 21 August 2020
Available online: 08 December 2020

Khomokhoana, P.K. and Nel, L. (2020). Decoding the underlying cognitive processes and related support strategies utilised by expert instructors during source code comprehension. *South African Computer Journal* 32(2), 232–257. <https://doi.org/10.18489/sacj.v32i2.811>

Copyright © the author(s); published under a [Creative Commons NonCommercial 4.0 License \(CC BY-NC 4.0\)](https://creativecommons.org/licenses/by-nc/4.0/). SACJ is a publication of the South African Institute of Computer Scientists and Information Technologists. ISSN 1015-7999 (print) ISSN 2313-7835 (online).

1 INTRODUCTION

Source code comprehension (SCC) is a core skill that many Computer Science (CS) students continue to struggle with (McCartney et al., 2013; Xie et al., 2018). SCC generally refers to the reading and interpreting of pieces of source code (Busjahn & Schulte, 2013; Lister et al., 2006). Some authors (Orlov et al., 2016; Praveen, 2016) describe it as a skill that requires efficient application of a series of complex cognitive processes. Due to the complex nature of SCC, a modelling process (Wood et al., 1976) – where instructors gradually guide their students in mastering these cognitive processes – could be instrumental in enabling students to perform tasks that were initially beyond their capacity. According to Middendorf and Pace (2004) – initiators of the Decoding the Disciplines (DtDs) approach which is gaining momentum worldwide at present – each academic discipline has its own distinctive set of mental operations that stakeholders follow when performing discipline-specific tasks. The explicit nature of these operations is, however, often so deeply buried in the unconscious minds of the discipline experts that it causes an expert blind spot (Pace, 2017). These blind spots occur when vital mental operations become so natural to the experts that they often omit crucial and even quite simple steps when explaining and modelling concepts and procedures to others (Nathan & Petrosino, 2003). Omission of such ‘hidden’ steps during instruction can lead to novices developing mental blocks (‘bottlenecks’) in mastering the steps involved in completing discipline-specific tasks (Pace, 2017).

The DtDs framework (Middendorf & Pace, 2004) includes a seven-step process that can be used to overcome specific student-learning bottlenecks. After identification of a specific bottleneck (Step 1), the disciplinary unconsciousness is systematically decoded in order to reveal the explicit steps followed by experts (e.g. instructors) when performing tasks related to the identified bottleneck (Step 2). These steps are then broken down into their component parts and each operation is modelled in a way that is understandable to students in order for it to be used to facilitate effective learning and understanding (Step 3). Students are then provided with opportunities to practise the modelled operations and get feedback on their efforts (Step 4). Throughout the process, specific strategies are employed to motivate students to follow the modelled operations (Step 5) and to assess whether they have mastered these operations (Step 6). In line with the principles of the Scholarship of Teaching and Learning (Felten, 2013), DtDs practitioners are encouraged to then share with other stakeholders what they have learned (Step 7).

Within the CS discipline, one of the most common and significant SCC bottlenecks identified (Khomokhoana & Nel, 2020; Lister et al., 2004) relates to students’ inability to reliably work their way through the long chain of reasoning required to comprehend a piece of source code. Instead of just using Step 2 of the DtDs process to uncover the explicit steps that experts follow in dealing with tasks related to this bottleneck, the complex cognitive nature of SCC (Praveen, 2016) has led us to first focus on identifying the underlying cognitive processes and uncovering the related support strategies utilised by experts to better execute these processes. If these cognitive processes are assumed as typical of the basic mental operations required for

efficient SCC, then these ways of thinking could point to important cognitive-related support strategies employed by experts during SCC. More explicit awareness of the cognitive processes and related support strategies instructors typically use to comprehend source code could help them to avoid their own blind spots when modelling efficient SCC strategies to their students. Such support strategies could also be used to help students to better execute the cognitive processes necessary for efficient SCC and ultimately support them to overcome related bottlenecks (Middendorf & Pace, 2004). Within the DtDs context, identifying important cognitive-related support strategies could serve as a precursor towards identifying the explicit steps followed by experts during SCC (which will ultimately be modelled to students in Step 3 of the DtDs process). This paper therefore attempts to answer the following two questions:

- What are the cognitive processes and related support strategies utilised by expert programming instructors during SCC?
- What does insight into these processes and strategies suggest in terms of important cognitive-related support strategies to be incorporated in the modelling of efficient SCC to students (as novice programmers)?

The remainder of this paper is organised as follows: Section 2 provides an overview of the basic cognitive processes involved in processing information and how these relate to SCC. In the discussion of the research design and method in Section 3, detail is provided about the selection of the experts and the SCC questions used in the decoding interviews. Important detail is also provided about the nature of the interviews and the way in which the interview questions were linked to cognitive processes in the data analysis. The presentation of findings (Section 4) takes place according to the main categories of cognitive processes recognised during the analysis of the transcribed interview data. As part of the discussion (Section 5), the experts' cognitive processes and related support strategies are summarised and important cognitive-related support strategies that should be emphasised by instructors while modelling efficient SCC strategies to their students are formulated. Conclusions are presented in Section 6.

2 BASIC COGNITIVE PROCESSES AND THE RELATION TO SOURCE CODE COMPREHENSION

Cognitive processes are defined as procedures that process all the information (multiple, complex or otherwise) human beings receive from their surrounding environment (Cognifit, nd.). The processing is done with the objective of transforming the information into easily manageable cognitive tasks (Newen, 2015). The basic cognitive processes discussed in the following sub-sections include attention, perception, memory, reading, speaking and listening as well as those processes related to reflective cognition. All of these processes are highly relevant to SCC and also to this study, as will be illustrated in the discussion.

2.1 Attention

Attention is a cognitive process in which certain items are selected (triggered by single or multiple stimuli) from a host of available possibilities at a certain point in time while doing something (Chandrika et al., 2017). When applying attention, one can use either the intensity or selectivity component (Pero et al., 2006). The intensity component enables a person to sustain concentration on one activity over time (sustained attention). The selectivity component enables a person to choose to focus on competing stimuli. This means that the attention may be divided and therefore not fully focused on the current activity.

During SCC, expert programmers often focus their attention on complex lines or sections of code (Itoh, 2019). These complex sections of code are likely to contain dynamic representations such as literals, comparisons, operators and keywords (Busjahn et al., 2011). Experts will typically identify these sections by quickly scanning through the code from top to bottom (Uwano et al., 2006).

Irrespective of type, attention is normally dependent on information that is relevant to the current task a person is performing (Preece et al., 2015). Van Someren et al. (1994) point out that in performing almost any task/activity, there will be irrelevant and distracting stimuli. As such, any individual involved in performing such a task should focus their attention by being conscious (e.g. recognise, differentiate, assemble things together, be assertive, be orientated and even suggest), alert, aware and responsive (reactive) in order to be successful (Oyebode, 2018).

2.2 Perception

Perception refers to the process of acquiring information from the world around us and transforming it into real experiences (Dhingra & Dhingra, 2011). Preece et al. (2015) point out that perception is a complex process that also involves other processes such as memory, attention, and language. Although perception can be used under normal circumstances, human beings have a tendency to use their perception when there is a breakdown in other cognitive processes (Sohlberg, 2000). Perception can also change while a specific task is performed. During this process, perceptual span increases when a person obtains useful information, while it decreases when encountering information that is difficult to comprehend (Orlov & Bednarik, 2017). Choi and Gordon (2013) argue that when perceptual span decreases, human beings will typically skip such troublesome information (e.g. words or text) and jump to sections that are not bothersome. While comprehending source code, programmers can form perceptions solely based on experience which could cause them not to consider all the stated facts in the problem being solved. Perceptions can also differ based on whether the programmer employs a bottom-up (Pennington, 1987), top-down (Brooks, 1983), knowledge-based (Letovsky & Soloway, 1986), systematic (Littman et al., 1987), micro (Letovsky, 1987), as-needed (Littman et al., 1987) or integrated (Von Mayrhauser & Vans, 1995) source code comprehension strategy. For example, while an as-needed strategy could cause programmers to only focus only on the code they perceive as relevant to the current task, programmers who choose to follow a

bottom-up strategy would be more likely to perceive every line of code as relevant.

2.3 Memory

Memory is a cognitive process that involves the recall of different kinds of knowledge that guide human beings to act or react in a specific way to certain stimuli (Preece et al., 2015). Knowledge can be recalled from either the long- or the short-term memory (Gage & Baars, 2018; Miller, 1956). It is, however, important to note that not all knowledge is stored in memory. A filtering process is used to decide what is processed and stored and what is not (Barkley, 2010). Since cognitive processes tend to overlap, the more attention a person pays to a certain aspect, the more likely it is that this aspect will be remembered (Preece et al., 2015). Programmers typically use strategies such as reading/re-reading specifications, thinking of possible test cases and reasoning aloud to enhance the memorability of concepts (Fitzgerald et al., 2005; Moore et al., 1997). Other strategies, such as highlighting or colouring some lines of code or text (Powell et al., 2004), writing comments (Scalabrino et al., 2016), pattern recognition (Fitzgerald et al., 2005) and making drawings or annotations (doodles) (Lister et al., 2004) are often utilised by programmers to readily and easily remember or determine the values of variables or other information without heavily engaging their memory.

2.4 Reading, speaking, and listening

Reading, speaking, and listening are three interrelated cognitive processes (Preece et al., 2015) that can be identified through facial expressions, vocal behaviour, verbal consent, pauses or segregates (e.g. 'hmm'), posture or stance, eye behaviour, hand gestures and head movements (Bonaccio et al., 2016; Knapp et al., 2014). A person can typically understand something well (e.g. a given piece of code) by using any one or a combination of these processes. An attempt to comprehend something that is written down and spoken requires more cognitive effort than just listening to it (Colter & Summers, 2014). However, many people prefer to listen, as they consider it the easiest method to comprehend something. In contrast, if something is written down, it is easier to re-read the information if it is not understood (Preece et al., 2015). Analogous to strategies used in other cognitive processes, programmers (when reading source code) will mark some lines of code (Powell et al., 2004), write comments (Scalabrino et al., 2016), draw illustrations (Lister et al., 2004) and/or read through the code multiple times (Moore et al., 1997) in an attempt to enhance their comprehension. During code reading, experienced programmers typically concentrate on the semantic features of the code, while less-experienced programmers tend to focus more on the syntactic features (Von Mayrhauser & Vans, 1995).

2.5 Reflective cognition

Planning, reasoning and decision making are interrelated cognitive processes that enable individuals to reflect on their cognition (Preece et al., 2015). During reflection, initial thoughts

and/or responses should be examined carefully before conclusions can be made. In doing so, a person will typically ask the following questions (Eisenführ et al., 2010; Herrmann, 2017; Uzonwanne, 2016):

1. What should I do (cognitive planning)?
2. What alternative courses of action do I have available (cognitive planning)?
3. Which alternative courses of action should I select to use (cognitive reasoning)?
4. Why should I use these (selected) alternative courses of action (cognitive reasoning)?
5. What are the consequences of using these (selected) alternatives (cognitive decision making)?

Inherently, questions 1 and 2 form part of cognitive planning. In addressing question 1, individuals actively and consciously engage their thought processes and use all resources available to them, such as discussions with others or using artefacts (for example, books, papers and the Internet) (Preece et al., 2015). This is done with the objective of better understanding the nature of the task in question in order to avoid ill-informed comprehension (Atherden, 2014). With regard to resources, Lister et al. (2004) recommend that a programmer should make some drawings or annotations (artefacts) in order to better comprehend source code. According to Hayes-Roth and Hayes-Roth (1979), question 2 is addressed in two stages. Firstly, a person decides in advance “a course of action aimed at achieving some goal” (pp. 275-76). Secondly, the execution of the plan is continuously monitored and guided to ensure success. This implies that the current course(s) of action can be revised over time based on new conditions encountered in the subsequent parts of the task at hand (e.g. SCC) (Guevara & Puche-Navarro, 2015).

Questions 3 and 4 essentially constitute cognitive reasoning. According to Evans (1993), an ability to arrive at the preferred alternatives involves some intelligent thinking. This implies that it is not the solution that is retrieved from the memory, but the relevant information. A person then needs to work out how best to apply it. Use of connectives such as and, if, or, all, some, none and not can be used as a basis in making some cognitive reasoning decisions in order to arrive at a solution to a problem (Knauff, 2007). It is important to note that during the cognitive reasoning process, a person (e.g. programmer) creates logical and systematic arguments and makes judgements based on these arguments (Gabor, 1976). Evaluating different arguments to decide which one is the best option involves actively and exhaustively processing information to ultimately decide on cost-effective courses of action for the task in question (Bauer et al., 2016).

Cognitive decision making is addressed by question 5. According to Preece et al. (2015), addressing this question involves working through different scenarios and gauging the good and bad points of each alternative. The 12 multiple-choice questions (MCQs) used in the study by Lister et al. (2004) are examples of situations where different scenarios – in this case missing

pieces of source code – are weighed against each other. Measures to mitigate unfavourable elements for each alternative are identified and documented at this stage. To make decisions, a person does not necessarily have to consider all the details included in the text or scenario. Instead, the focus can be placed on only a few key indicators (Preece et al., 2015). By doing so, a person provides justification for all decisions arrived at (Herrmann, 2017).

Planning, reasoning and decision making can be regarded as steps in the process of solving a problem. This process is characterised by certain actions that a person performs prior to and throughout solving a problem. Due to the cognitive nature of problem solving, a person should continuously engage and stimulate their thought processes when solving a problem (Jones, 2007). To be successful in problem solving, Frederick (2005) recommends that people should have “the ability or disposition to resist reporting the response that first comes to mind” (p. 35). This emphasises the argument that after identifying a solution to a programming problem, the solution should be evaluated, implemented and re-evaluated. These stages should also be revisited frequently during the iterative implementation of the solution and the discovery of more knowledge that was not apparent to the programmer (Loksa et al., 2016).

Having provided some background on what constitutes the aforementioned cognitive processes, the next section discusses the research design and the procedure that was followed to identify the underlying cognitive processes and uncover the related strategies employed by expert programming instructors in this study.

3 RESEARCH DESIGN AND METHOD

The design of this study was narrative in nature and focused on the ‘asking questions’ data collection strategy, as described in Plowright’s Frameworks for an Integrated Methodology (FraIM) (Plowright, 2011). A case study was deemed the most appropriate data source management strategy, since only a small number of participants would be used. The population included CS instructors from a selected South African higher education institution. The sample consisted of five instructors who were purposefully (Cooper & Schindler, 2013) selected based on the fact that they were all experienced CS instructors who had been involved in teaching programming to novices (as part of CS1 and/or CS2 courses) for at least three years. Two of the participants (P1 and P4) had more than 14 years of experience in this regard, while P2 and P3 had between five and nine years of similar experience. Except for P5, all the other participants worked as industry programmers for at least four years and they were all, to some extent, still involved in private programming consultancy work. This sample can also be regarded as convenient (Patton, 2015), since the selected participants were in proximity to the principal researcher (the first author) and could therefore be reached easily.

3.1 Data collection

As part of the ‘asking questions’ data collection strategy, primary data was collected by means of decoding interviews (Middendorf & Pace, 2004) with experts – as distinctively employed

in step 2 of DtDs. This data was supplemented by a short questionnaire and observations. Experts are characterised as individuals who perform critical thinking tacitly and implicitly in their own disciplines (Middendorf & Pace, 2004; Pace, 2017). As such, the aim of the decoding interviews was to identify the underlying cognitive processes and related strategies that expert programming instructors would utilise in order to accomplish tasks that students find difficult to execute. A decoding interview is typically conducted by at least two interviewers (Pace, 2017, p. 39). Given the format of a decoding interview, a single interviewer might get lost in the details, while two minds could better keep the interviewing process on track (Middendorf & Shopkow, 2018). During the interview process, both interviewers should be able to verbalise their thinking, challenge the explanations given by the interviewees and summarise their thinking back to the interviewees on an abstract level (Shopkow et al., 2013). For this reason, Middendorf and Pace (2004) describe the interview process as the most intellectually demanding of all the DtDs steps.

Since members of the same discipline have a tendency to share common expert blind spots, Pace (2017) recommends that the second interviewer should ideally come from outside the discipline. The second interviewer is then more likely to see when a specific mental step has not been fully explained. However, given the complex nature of the cognitive processes involved in SCC and the authors' inability to find a suitable person with relevant decoding interview experience from outside the discipline, an alternative arrangement had to be made. For the decoding interviews in this study, the principal researcher acted as the principal interviewer, with the support of a non-teaching CS researcher who had some decoding interview experience as the second interviewer.

3.2 Data collection procedure

All participants completed an informed consent form (as stipulated in the ethical clearance authorisation granted by the institution) before participating in the decoding interview. The proceedings of each interview were audio recorded with the permission of the participant. In each interview, the participants were first asked to explain the process they would go through when they needed to comprehend any given piece of source code. Whenever the interviewers felt that the participants were not clearly verbalising all their mental operations, one of them would intervene with a probing question. After about 30 minutes, a specific SCC question was presented to the participants, asking them to verbally illustrate the general SCC process they had just explained in answering this question. Where necessary, further probing questions were asked. At the end of the interview session, the participants completed a short questionnaire to provide basic demographic data and information regarding their programming and teaching experience. The principal interviewer also made observation notes throughout each interview.

Although the original plan was to include three SCC questions in this part of the decoding interview, a pilot of the entire data collection procedure revealed that it would take too long and that sufficient data could be collected if just one question was used. Question 6 (see

Figure 1) from the original set of 12 MCQs developed by the ITiCSE 2004 working group for their multi-national study of reading and tracing skills in novice programmers (Lister et al., 2004) was therefore selected. This question was identified as the second most challenging question in Lister et al.'s study (2004). While the most challenging question (Question 12) mostly focuses on arrays, Question 6 covers a variety of programming concepts (including Boolean variables, for loops, array indexes, and return statements to terminate a for loop). In answering Question 6, the missing piece of source code had to be identified from the five given options (note: the correct answer is option B). The only change made to the question was to convert it from the original Java to C# (the programming language that all the participants were familiar with). The line numbers as illustrated in Figure 1 were not part of the question given to participants and are only included here for ease of reference in the results discussion to follow.

3.3 Data analysis

Following the decoding interview proceedings, a narrative data analysis approach as suggested by Creswell and Creswell (2017) was used to transcribe the audio recordings made during the decoding interview sessions and to analyse the data. After transcribing the data, it was cleansed by searching for faults and repairing them (Chu et al., 2016). As the discussions were open-ended, the transcripts also contained some illogical and repeated statements. It was therefore decided to use fuzzy validation (Parcell & Rafferty, 2017), which allowed the researchers to make some corrections to the data if there was a close match or known answer. After this, the researchers familiarised themselves with the data (Liamputtong, 2009) by listening and re-listening to the audio recordings numerous times, as well as intensively reading and re-reading the transcripts. This helped to decide on a coding plan where the analysis was guided by the data as it relates to the first research question. At this point, the five validated transcripts were imported into NVivo for Windows, Version 12 for further analysis. Codes were then developed (by creating several nodes) for each cognitive process recognised in the data.

As suggested by Saldaña (2016), the data was then coded by highlighting and/or underlining sections/passages (e.g. words/keywords, sentences, paragraphs) from which cognitive processes could be extracted (under the guidance of the theoretical guidelines as identified from the literature). The developed codes were then populated by moving the necessary text into them. Consequently, some themes started to emerge which revealed important information about the data set in relation to the first research question (Braun & Clarke, 2006). Continuing with this process led to the emergence of recurrent themes. Finally, NVivo was used to generate frequencies of occurrence for each of the developed themes.

The following method `isSorted` should return `true` if the array is sorted in ascending order. Otherwise, the method should return `false`:

```

1. public static bool isSorted (int[] x)
2. {
3.     //missing source code goes here
4. }
```

Which of the following is the missing source code from the method `isSorted`?

- a)


```

5. bool b = true;
6. for (int i = 0; i < x.Length - 1; i++)
7. {
8.     if (x[i] > x[i + 1])
9.         b = false;
10.    else
11.        b = true;
12. }
13. return b;
```
- b)


```

14. for (int i = 0; i < x.Length - 1; i++)
15. {
16.     if (x[i] > x[i + 1])
17.         return false;
18. }
19. return true;
```
- c)


```

20. bool b = false;
21. for (int i = 0; i < x.Length - 1; i++)
22. {
23.     if (x[i] > x[i + 1])
24.         b = false;
25. }
26. return b;
```
- d)


```

27. bool b = false;
28. for (int i = 0; i < x.Length - 1; i++)
29. {
30.     if (x[i] > x[i + 1])
31.         b = true;
32. }
33. return b;
```
- e)


```

34. for (int i = 0; i < x.Length - 1; i++)
35. {
36.     if (x[i] > x[i + 1])
37.         return true;
38. }
39. return false;
```

Figure 1: SCC question used in decoding interview

4 FINDINGS AND INTERPRETATION

Given the large amount of data collected during the decoding interviews, this paper focuses on data collected during the time when the participants were tackling the given SCC question (see Figure 1). The discussion in the following sub-sections focuses on the five cognitive process categories that were observed, as well as evidence of their occurrence.

4.1 Reflective cognition

The category for reflective cognition processes had the highest number of occurrences (73). Participant 1 (P1) employed cognitive strategies extensively in this category, with 25 occurrences. Findings on the three cognitive processes constituting this category as well as the all-encompassing problem-solving process are discussed next.

4.1.1 Planning

During the cognitive planning process, a person needs to decide on the course of action to follow in order to arrive at a solution to a problem (Hayes-Roth & Hayes-Roth, 1979). P3 demonstrated some elements of planning, as is evident from the following excerpt:

You have already told me what the output should be. So now I have a question: What is it supposed to do? But there is a mistake, what is missing that would create the correct output? Now I know I need to look at this code for an error. The first thing I need to do is figure out which thing is doing what and how they are working together, and then I can find the mistake – unless the mistake is something obvious, in which case I can quickly find it.

It can be seen from this excerpt that P3 first familiarised himself with the problem specifications so that he would know what was expected of him. The problem statement gave him an idea of what the missing source code should do. He also inferred that there might be errors in some of the source code fragments that would make them fail to produce the desired output if they were to be placed in the `isSorted` method. Further, by recognising that some pieces of code may have conditions or errors that could disqualify them from being the correct piece of missing code, P3 was already deciding on what he would do to ultimately arrive at the correct piece of missing code. Moreover, this strategy could help him spend considerably less time to ultimately get to the final answer, as he seemed to be applying efficient planning.

4.1.2 Cognitive reasoning

The cognitive reasoning process requires a person to integrate all the information at their disposal in order to arrive at a solution to a problem (Evans, 1993). P1 exhibited this type of reasoning when he started by focusing on the opening statement of the question and the

method signature in Line 1 (see Figure 1). He examined and read the various aspects of the question and specifically employed a cognitive reasoning process based on what he found. For example, to decide on the parameters of the `isSorted` method, he read, interpreted and comprehended the method signature of the question. This, in turn, enabled him to apply some reasoning throughout the tracing task process. The likelihood of identifying the correct missing source code could have been low if this method signature was not well understood.

Furthermore, P1 made logical and systematic arguments (Knauff, 2007) as he proceeded with the interpretation of the question. Evidence of this argument is contained in the following excerpt:

The `isSorted` method will receive an array of integers. It will receive it in `x` and now I will need to look at each one of these options here to see what goes in the missing code.

Referring to ‘options’ in the above excerpt is already an indication that P1 was working towards an elimination strategy (Fitzgerald et al., 2005) to remove incorrect options.

4.1.3 Decision making

During the decision-making process, a person will focus on those problem details that can improve understanding of the problem so that well-informed decisions are made in tackling the problem at hand (Preece et al., 2015). For example, P1 made a strategic decision based on the length of the question:

So, in such a case, when it is a long question like this, I would quickly scan through the options to see which one is most probably going to be the correct one. And then I start with that one instead of doing the first one and run through the entire thing.

As can be seen from this excerpt, P1 also employed an effective ‘quick scan’ strategy as suggested by Bauer et al. (2016). This strategy allowed him to start working towards an answer at an opportune place, hence expediting the process of finding the right answer.

As another example of decision making, P3 familiarised himself with the actual code included in the question by reading through the pieces of source code in order to determine the meaning of each statement. In this regard, he said:

Now I pick a set of inputs 0, 1, 2, and I see that we are obviously looping through the array. So, I can see that we are stopping at the second-last element. This is the case again where I do not need to look through this in great detail, because I can see that thing. That is easy access to my array, because you said it there. It is my array length minus 1, which means the second-last element and we are going less than that. So, I understand it to be saying that we are going through each element in the array up until the second-last element, and I do not have to look at that loop again.

From the above excerpt, it is also interesting to note that P3 decided to use some test cases or sample values to identify the limits of the array in question. Moreover, recognising that lines 6, 14, 21, 28 and 34 were identical, eliminated the need for P3 to interpret each one of them. This means that he interpreted the statement on line 6 and applied that interpretation to all similar statements.

4.1.4 Problem solving

Given the close relation between planning, reasoning, and decision making as part of the problem-solving process (Preece et al., 2015), it is worthwhile to consider how one of the experts utilised each of these cognitive processes as part of her problem-solving process.

While reading through the problem specification, P2 started to formulate her problem-solving strategy by identifying what she was asked to do and what her first action should be. The following excerpt illustrates her cognitive planning process:

So I first try to understand the question (reading the question)... otherwise the method should return false, right! So which one of the following is the missing source code from the method isSorted?... So what I am going to do is to scan through each of my options quickly and see if I can eliminate others very quickly.

Her main aim with this strategy was to quickly identify the ‘obvious’ incorrect options so that she would not have to spend unnecessary time on in-depth interpretation of those code segments.

In her scanning of the initial lines of code (reading the code sequentially), she said: “All of them [alternative answers] are returning values, all of them are determining Booleans. So, I can’t eliminate an alternative based on that”. In this regard, she was using cognitive reasoning to formulate a set of elimination criteria. She further pointed out that these criteria were based on both the question and what she saw in the given code fragments. After considering the above, she continued her reasoning process to examine additional aspects of the code alternatives: “Can I eliminate one? Yes or No? If I can’t eliminate one, now I start looking in more detail”. This examination of the possible code options eventually led her to apply her decision-making skills when she tentatively marked option B as the possible missing piece of code: “So I will mark B as a possible solution. Because at the moment, without going really in depth and using a test case, it looks to me like it will work”. However, instead of settling for option B right away, she continued as follows: “So I will just go to B and I will check it again”. This is illustrative of the thoroughness strategy, as suggested by Fitzgerald et al. (2005). This is the strategy where a person, upon initially recognising an answer, checks further for the correctness or incorrectness of answers just to be convinced of the final answer.

As illustrated in this problem-solving example, P2 followed an informed problem-solving process that enabled her “to resist reporting the response that first came to her [sic] mind” (Frederick, 2005, p. 35). Moreover, from the observation notes and all the steps that P2 followed in answering the question, it could be seen that she had mastered the problem-solving skill.

4.2 Attention

The attention process is characterised by selecting single or multiple options from a host of available possibilities (Chandrika et al., 2017). In this study, 52 occurrences of using the attention cognitive process were observed in the participants. P1 employed this category the most, with 21 occurrences. As an indication of paying attention, P1 uttered the following statement: “Now I see that B and E do not have that declaration and it will only do a comparison in the if statement and then return a specific value”. By using an attention cognitive process, he was therefore able to identify that only three of the options (A, C and D) included a declaration for the Boolean variable *b*.

P2 used her attention process to realise that all five options to the question were returning values, as is evident from the following excerpt: “All of them are returning values, all of them are determining Booleans”. As the cognitive processes tend to overlap (Preece et al., 2015), she immediately switched from attention to decision making (a reflective strategy) by saying, “so I cannot eliminate an option based on that”.

As an indication that P3 was paying attention while reading the question specifications, he said:

First thing, I must make sure that I understand the question. So, you have helped me a bit by boldfacing some words. So, I will read the question focusing only on the boldfaced words. Then I will read it again and I will boldface in my mind some other words such as array, method, sorted – so those are words that immediately come to mind. Other words glue everything together.

It should be noted that the ‘boldfaced words’ referenced by P3 were actually a different font face used to ensure that code statements and values would stand out from the normal sentence text. With his attention drawn to these words, P3 identified those as important sections (Crosby et al., 2002). He proceeded further by identifying even more ‘keywords’ (e.g. array, method and sorted) to help focus his attention while tackling the problem.

4.3 Reading, speaking and listening category

As part of the decoding interview, participants had to listen to the interviewers’ questions and verbally explain their SCC processes. For the purpose of this discussion, these listening and speaking actions are not regarded as part of the experts’ natural SCC strategies. We instead focus on the 17 other occurrences of these cognitive processes as observed in the participants. These actions included body movements (hand, eye, and head), facial expressions and the utterance of some words (Bonaccio et al., 2016; Knapp et al., 2014). P1 applied this category the most, with six occurrences.

While reading the specifications, P1 focused on the semantic processing of the method signature (see Line 1) to determine that `isSorted` was a static method of a `bool` type which would receive an array of integers. Similar to our other experts, he realised the importance of the method signature and chose to focus on semantics instead of syntax. This is in contrast to

a novice programmer who would typically rely more on syntactic aspects or even choose to ignore the method signature and its parameters completely (Von Mayrhauser & Vans, 1995). As further evidence of semantic processing, P3 referred to the specifications as a 'rule' that guides the entire process of working through the comprehension task and arriving at the desired answer. He explicitly shared the information that he gathered from reading the problem specifications. He also advised that it is recommendable to read problem specifications at least twice, as is suggested by Moore et al. (1997).

P4 regarded speaking and listening as fundamental strategies in his way of doing things (including code comprehension): "I always speak aloud! My wife came to me this morning. She said: 'You are getting mad'. She always hears me talking to myself". He further confirmed that even for general reasoning, he tends to reason aloud. He argued that this technique helps him to go through the logic fast and also ensures that he does not skip the logic steps as he is listening to himself. This combination of cognitive reading, speaking and listening also allows him to commit the information to memory (Preece et al., 2015).

4.4 Memory

The memory cognitive process requires the use of support strategies that make it easier for a person to remember and/or recall values or knowledge when necessary (Preece et al., 2015). Many of the attention, reading, speaking and writing processes as discussed above, also helped the experts in this study to enhance their memory. Based on this notion, 11 occurrences of this cognitive process were identified.

During the decoding interview, P2 was observed making some pen-holding hand gestures as if she was writing computations in the air. In response to a question asking if she was "passing values in her head as she went through the code", she replied "Yes" – confirmation that she was using her working memory. Essentially, it is not easy to remember things that are not written down. Being compelled to do so can lead to a working memory overload (De Jong, 2010). As further evidence that P2 was doing some comparisons in her mind, she was observed comparing the last two elements of the sample array she created herself. When asked whether she considered the previous values, she said: "I compared them in my mind". P3 also did not show the test cases he used. However, when asked if he was doing "the test cases in his mind", he admitted that he was.

The behaviour exhibited by P2 and P3 is interesting, because Wiedenbeck (1985) observed that experienced programmers required less mental attention. However, experts in this study mostly kept things in their working memory. A possible explanation for this type of behaviour could be that these things were still manageable (i.e. within the number 5 plus or minus 2 items according to Miller (1956)). When the information became too much to keep in memory, the experts resorted to other strategies (Bransford et al., 2000) to help them remember values and/or keep track of the program logic.

Due to the prominence of the integer array x in Q6, most of the participants used some type of doodle (Lister et al., 2004) to represent the array elements. Others just wrote down the

values of variables (e.g. '2, 7, 5'), plainly as an indication that they were using these arbitrary values as test cases to help them determine outputs for the given code fragments. To enhance his memory, P5 resorted to pattern recognition (Fitzgerald et al., 2005) after realising that the for loops in all five alternative fragments of code (Lines 6, 14, 21, 28 and 34) were identical. During the interview, P5 was asked the following question by interviewer 2: "Did you use the pattern from option A and apply it to option B?" In response, P5 said: "Yes, I did not have to check whether this condition makes sense again". This confirms that he was using some recall of things or actions he did before. He went on to apply the same pattern(s) he had seen in the previous options, and to apply them in subsequent instances. Inherently, these strategies helped the expert programmers to easily recall variable values and/or important information when they wanted to use it.

4.5 Perception

Perception is often used when other cognitive processes fail to help us make sense of what we are trying to understand or achieve (Sohlberg, 2000). With reference to this notion, a total of eight occurrences of the application of the perception process were identified with the participants. However, in all of these instances the participants' perception caused them to focus on issues that were not directly related to solving the given SCC task (for example, the coding convention, the use of return to break out of the for loop and the level of difficulty of the task). Consequently, the identified perception strategies were regarded as unrelated to the aim of this paper and were therefore not considered for inclusion.

5 STRATEGIES TO SUPPORT COGNITIVE PROCESSES DURING SOURCE CODE COMPREHENSION

During the SCC task, the expert programmers mostly relied on four cognitive processes to efficiently comprehend the provided source code: reflective cognition, attention, reading and memory. For each of these cognitive processes, the experts utilised very specific support strategies to help them to better execute these cognitive processes. In the sub-sections to follow, these cognitive-related support strategies are first summarised and then the most important of these strategies are extracted.

5.1 Reflective cognition

The experts interviewed in this study often asked themselves guiding questions, as also suggested by literature (Eisenführ et al., 2010; Herrmann, 2017; Uzonwanne, 2016). These questions allowed them to form logical arguments to solve the given SCC problem. Furthermore, they did not seem to take for granted any information included as part of a problem. Instead, they used every single piece of information to formulate their logical and systematic arguments

(Knauff, 2007). In this regard, it was important for them to not only work out the meaning of each piece of code, but also how all the individual parts were linked together.

The expert programmers spent ample time familiarising themselves with the task requirements by intensively reading (and even re-reading) the question. They clearly wanted to ensure that they fully comprehend the problem before attempting to solve it (Atherden, 2014). This ultimately enabled them to decide on the best strategies to use in tackling the given problem. Additionally, they employed time-saving reasoning strategies such as ‘quick scan’ (reading through the code quickly just to get an overview) (Bauer et al., 2016) and purposive elimination of identical code segments (see Section 4.1.3) in an attempt to reduce the time and effort they needed to solve a problem. However, their main focus was not to arrive at the answer as quickly as possible, instead, they patiently followed appropriate and robust strategies, as suggested by literature (Fitzgerald et al., 2005; Frederick, 2005), to exhaust all possibilities to verify and ensure the correctness of their final answer (see Section 4.1.4). In most cases, this was achieved by double-checking the logic they followed to arrive at an answer.

5.2 Attention

In solving SCC problems, the experts who participated in this study were seen to be paying attention in the true sense. As a result, they were able to identify some aspects (e.g. similar lines of code, used data structures, etc.) that readily informed them on how to best tackle a given problem. In addition, these experts were observed to possess a skill that helped them to immediately switch to other strategies or alternatives (Sohlberg, 2000) based on encountered information. This helped them to avoid getting stuck in certain areas of the source code or problem description. As a result, there were no signs of frustration, discomfort, and disorganisation (Khomokhoana & Nel, 2020) observed with the experts while solving the given SCC task.

Similar to the findings of Busjahn et al. (2011), the experts paid more attention to complex code statements and functional details than to other simple or superficial details. By focusing their attention, they were able to temporarily ignore non-vital details at opportune moments (as suggested by Preece et al. (2015)). Moreover, cases of switching from one cognitive process to another were observed in the experts. A typical example was when one expert (P2) switched from attention to decision making (see Section 4.2). Switching attention in general seemed to have been helpful to the experts throughout their SCC process.

5.3 Reading

In reading the given code comprehension scenarios, the experts made sure that they understood what they were reading by re-reading the text of such scenarios, even if they thought they understood it. It was interesting that the experts who could be considered the most experienced made sure of this and even emphasised the importance of doing this. As reading or re-reading is observed through eye or pen or hand movements (Bonaccio et al., 2016; Knapp

et al., 2014), our experts even mentioned that they were re-reading the details or some code fragments to confirm their initial understanding. They also interpreted and re-interpreted the various components of the scenario, implying that they did not automatically rely on their very first interpretations. This practice seems to have happened effortlessly with the experts, suggesting that their brains were already ‘wired’ or prepared for such practices.

5.4 Memory

From the observations made (see excerpts in Section 4.4), the experts seemed to possess innate knowledge that they applied if they did not note or write things down (Bransford et al., 2000). Otherwise, they wrote down everything they believed necessary or might be required during the SCC process. This helped them to easily update current variable values as and when necessary without straining their memory (Xie et al., 2018). Another strategy the experts used was to consider limited scenarios at a time (e.g. comparing a worst- and best-case scenario). By minimising the amount of details kept in their memory, they did not necessarily have to write things down as they did not overload their working memory. The experts also indicated that they used support strategies such as thinking aloud to help them remember certain information. Of particular interest was the expert (P4) who indicated that think-aloud was a fundamental technique he uses daily in almost everything he does (see Section 4.4).

5.5 Cognitive-related support strategies for SCC

Given the complex cognitive nature of SCC (Praveen, 2016) and novice programmers’ continuous inability to reliably work their way through the long chain of reasoning required to comprehend a piece of source code (Khomokhoana & Nel, 2020; Lister et al., 2004) (the identified bottleneck), it is vital that CS educators are made aware of the specific support strategies they, as experts, typically utilise to execute the cognitive processes necessary for efficient SCC. Table 1 presents 17 cognitive-related support strategies that were extracted from the findings of this research study and the supporting literature. In addition to the relevant reading, speaking/listening, attention and memory cognitive processes, each of these support strategies are also related to one or more of the reflective cognition processes (planning, cognitive reasoning and/or decision making) required for problem solving. Problem solving (as a vital element of efficient SCC) is one of the skills not typically taught to students explicitly, as instructors tend to concentrate more on core course content than on external aspects of the learning process (Loksa et al., 2016).

Table 1: Cognitive-related support strategies

Cognitive-related support strategies for efficient SCC	Cognitive processes						
	Planning	Reasoning	Decision making	Reading	Speaking/listening	Attention	Memory
Read through the problem specification at least twice (or until you understand what you are asked to do)	✓			✓			
While reading through the specifications, mark/highlight important words	✓			✓		✓	
Do a quick scan of the provided source code. Mark important sections AND complex sections of code	✓			✓		✓	
For any complex sections of code, first make sure that you understand the meaning/working thereof BEFORE you continue to solve the problem	✓			✓			✓
Identify any code segments that appear more than once (repeated code)	✓					✓	
Read through all the provided code at least twice to make sure that you understand everything	✓			✓			
Write down any additional information that might be relevant in solving the problem	✓					✓	✓
Identify at least two strategies you could follow to solve the problem	✓					✓	
Compare the possible strategies and select ONE that you think will work best to solve the problem		✓	✓				
Do not be afraid to adapt or change your strategy if it is not working	✓	✓	✓				
For questions with multiple answer options, first focus on evaluating options that seem ‘more correct’ at first glance. Leave the ‘possibly incorrect’ options for later (if none of the ‘more correct’ options turn out to be the correct answer)		✓	✓			✓	
If you get stuck, consider the wider context in which the code or piece of information appears/is used	✓	✓	✓			✓	
Define and use your own test case values (if not provided)		✓					✓
While tracing through the code, keep track of changes in variable values on paper		✓					✓
Draw a diagram to visualise your understanding of the program logic		✓					✓
Reason aloud while working on solving the problem (if possible)		✓			✓		✓
Once you have arrived at an answer, double-check your reasoning to confirm the correctness of your answer		✓	✓				

Although these support strategies might be regarded as ‘straightforward’, ‘simple’ or even ‘obvious’, there is no use in just mentioning these strategies in passing to students and taking it for granted that they will follow these strategies. Ultimately, these support strategies need to be modelled in a way that is understandable to students so that they can be used to facilitate effective learning and understanding of SCC (during Step 3 of the seven-step DtDs process). Thereafter, students should be provided with opportunities to practise the modelled operations and get feedback on their efforts (DtDs Step 4). At the same time, the educator will need to employ specific strategies to motivate students to follow the modelled operations (DtDs Step 5) before the students’ mastery of these operations can be assessed (DtDs Step 6). The implementation of such a teaching and learning strategy will undoubtedly require a necessary change in practice from educators – forcing them to move out of their ‘comfort zones’. The following DtDs-related quote from Hutchings et al. (2011) serves as a stark reminder of how comfortable academics often are with their current ways of doing:

Disciplinary practices – ways of reading, thinking, questioning, investigating – can become the water one swims in: invisible, just assumed, and a pretty darn nice temperature most days of the week.

In this regard, the DtDs framework provides a platform for educators to not only reflect on the effectiveness of their own classroom practices, but also to rethink the ways in which they model their own expert ways of thinking to their students.

6 CONCLUSIONS

In focusing on Step 2 of the DtDs framework, this study primarily utilised decoding interviews to identify four categories of cognitive processes (attention; memory; reading, speaking and listening; and reflective cognition) and uncover related support strategies that are essential for efficient SCC. The 17 cognitive-related support strategies that were identified in this research study (through the use of extensive probing questions as well as focused observations during the decoding interviews) can be regarded as indicative of simple (yet very useful) strategies that are often not explicitly and/or continuously modelled to students (Middendorf & Pace, 2004) – thereby revealing an expert blind spot towards the nature and importance of such essential cognitive-related support strategies in guiding novice programmers during SCC tasks. The way in which such a discipline-specific task is modelled to students should include all the mental steps followed by the instructor (the expert), including the basic or even assumed trivial steps. By creating awareness regarding all the cognitive processes and related support strategies required for efficient SCC, the authors hope to, firstly, make other CS experts and instructors more aware of their own potential SCC blind spots. Secondly, the uncovered support strategies highlight mental operations that should be explicitly modelled to and practised by students. This could help students to better execute the cognitive processes necessary for efficient SCC and ultimately support them to overcome related SCC bottlenecks.

This study illustrated the application and potential value of a DtDs decoding process in CS education. There are, however, numerous variations within each of the steps that can still be exploited within the discipline. Decoding interviews, for example, is just one of several methods that can be utilised in Step 2 of the DtDs process to systematically decode the disciplinary subconscious of experts (Middendorf & Shopkow, 2018). All of the proposed methods are, however, aimed at revealing the explicit steps that an expert would follow when performing tasks related to an identified bottleneck (Middendorf & Pace, 2004). In this study, the complex cognitive processes required to comprehend source code (Praveen, 2016) have led the authors to instead focus (in the first part of the decoding interview) on exposing specific strategies that the experts would follow to support these cognitive processes. The nature of the SCC process, however, created an opportunity for the inclusion of a novel additional component in the second part of the decoding interviews. The addition of a think-aloud component provided the authors with a unique opportunity to directly observe the experts' actual execution of the identified cognitive processes during a real SCC task. This provided the authors with even more insight into the nature of the cognitive processes and related support strategies required for efficient SCC. Knowledge of the cognitive-related support strategies (as presented in Table 1) could therefore serve as a precursor towards exposing the explicit steps followed by experts during SCC in future research as part of the remaining steps of the DtDs framework.

References

- Atherden, C. (2014). *Can do problem solving Year 1*. Nelson Thornes.
- Barkley, E. (2010). *Student engagement techniques: A handbook of college faculty*. Jossey-Bass.
- Bauer, T., Erdogan, B., Short, J. & Carpenter, M. (2016). *Principles of management*. Flat World Management.
- Bonaccio, S., O'Reilly, J., O'Sullivan, S. L. & Chiocchio, F. (2016). Nonverbal behavior and communication in the workplace: A review and an agenda for research. *Journal of Management*, 42(5), 1044–1074. <https://doi.org/10.1177/0149206315621146>
- Bransford, J., Brown, A. & Cocking, R. (2000). *How people learn: Brain, mind, experience and school*. National Academy Press.
- Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6), 543–554. [https://doi.org/10.1016/S0020-7373\(83\)80031-5](https://doi.org/10.1016/S0020-7373(83)80031-5)
- Busjahn, T. & Schulte, C. (2013). The use of code reading in teaching programming. *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, 3–11. <https://doi.org/10.1145/2526968.2526969>
- Busjahn, T., Schulte, C. & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, 1–9. <https://doi.org/10.1145/2094131.2094133>

- Chandrika, K., Amudha, J. & Sudarsan, S. (2017). Recognizing eye tracking traits for source code review. *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8. <https://doi.org/10.1109/ETFA.2017.8247637>
- Choi, W. & Gordon, P. (2013). Word skipping during sentence reading: Effects of lexicality on parafoveal processing. *Attention, Perception & Psychophysics*, 76, 201–213. <https://doi.org/10.3758/s13414-013-0494-1>
- Chu, X., Ilyas, I., Krishnan, S. & Wang, J. (2016). Data cleaning: Overview and emerging challenges. *Proceedings of the 2016 International Conference on Management of Data*, 2201–2206. <https://doi.org/10.1145/2882903.2912574>
- Cognifit. (nd.). Cognition and cognitive science [Last accessed 05 Nov 2020]. <https://www.cognifit.com/cognition>
- Colter, A. & Summers, K. (2014). Low literacy users. In J. R. Bergstrom & A. J. Schall (Eds.), *Eye tracking in user experience design* (pp. 331–348). <https://doi.org/10.1016/B978-0-12-408138-3.00013-3>
- Cooper, D. & Schindler, P. (2013). *Business research methods* (6th ed.). McGraw-Hill Education.
- Creswell, J. & Creswell, J. (2017). *Research design: Qualitative, quantitative and mixed methods approaches* (5th ed.). Sage Publications.
- Crosby, M., Scholtz, J. & Wiedenbeck, S. (2002). The roles beacons play in comprehension for novice and expert programmers. *Proceedings of the 14th Workshop of the Psychology of Programming Interest Group*, 58–73.
- De Jong, T. (2010). Cognitive load theory, educational research, and instructional design: Some food for thought. *Instructional Science*, 38, 105–134. <https://doi.org/10.1007/s11251-009-9110-0>
- Dhingra, M. & Dhingra, V. (2011). Perception: Scriptures' perspective. *Journal of Human Values*, 17(1), 63–72. <https://doi.org/10.1177/097168581001700104>
- Eisenführ, F., Weber, M. & Langer, T. (2010). *Rational decision making*. Springer-Verlag.
- Evans, J. S. B. T. (1993). The cognitive psychology of reasoning: An introduction. *The Quarterly Journal of Experimental Psychology*, 46(4), 561–567. <https://doi.org/10.1080/14640749308401027>
- Felten, P. (2013). Principles of good practice in SoTL. *Teaching and Learning Inquiry*, 1(1), 121–125. <https://doi.org/10.2979/teachlearninqu.1.1.121>
- Fitzgerald, S., Simon, B. & Thomas, L. (2005). Strategies that students use to trace code: An analysis based in grounded theory. *Proceedings of the First International Workshop on Computing Education Research*, 69–80. <https://doi.org/10.1145/1089786.1089793>
- Frederick, S. (2005). Cognitive reflection and decision making. *Journal of Economic Perspectives*, 19(4), 25–42. <https://doi.org/10.1257/089533005775196732>
- Gabor, P. (1976). Management theory and rational decision making. *Management Decisions*, 14(5), 274–281.
- Gage, N. & Baars, B. (2018). *Fundamentals of cognitive neuroscience: A beginner's guide*. Academic Press.

- Guevara, M. & Puche-Navarro, R. (2015). The emergence of cognitive short-term planning: Performance of preschoolers in a problem-solving task. *Acta Colombiana de Psicología*, 18(2), 13–27. <https://doi.org/10.14718/ACP.2015.18.2.2>
- Hayes-Roth, B. & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science*, 3(4), 275–310. https://doi.org/10.1207/s15516709cog0304_1
- Herrmann, J. W. (2017). Rational decision making. In N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri & J. L. Teugels (Eds.), *Wiley StatsRef: Statistics Reference Online* (pp. 1–9). John Wiley & Sons.
- Hutchings, P., Huber, M. & Ciccone, A. (2011). *The scholarship of teaching and learning reconsidered: Institutional integration and impact*. Jossey-Bass.
- Itoh, T. (2019). Towards generation of visual attention map for source code. *Proceedings of the 11th annual conference organized by Asia-Pacific Signal and Information Processing Association (APSIPA)*. <http://arxiv.org/abs/1907.06182>
- Jones, M. (2007). The redesign of the delivery of an introductory programming unit. *Innovation in Teaching and Learning in Information and Computer Sciences*, 6(4), 169–182. <https://doi.org/10.11120/ital.2007.06040169>
- Khomokhoana, P. J. & Nel, L. (2020). Decoding source code comprehension: Bottlenecks experienced by senior computer science students. In B. Tait, J. Kroeze & S. Grüner (Eds.), *ICT Education. SACLA 2019. Communications in Computer and Information Science* (pp. 17–32). https://doi.org/10.1007/978-3-030-35629-3_2
- Knapp, M., Hall, J. & Horgan, T. (2014). *Nonverbal communication in human interaction*. Cengage Learning.
- Knauff, M. (2007). How our brains reason logically. *Topoi*, 26(1), 19–36. <https://doi.org/10.1007/s11245-006-9002-8>
- Letovsky, S. (1987). Cognitive processes in program comprehension. *Journal of Systems and Software*, 7(4), 325–339. [https://doi.org/10.1016/0164-1212\(87\)90032-X](https://doi.org/10.1016/0164-1212(87)90032-X)
- Letovsky, S. & Soloway, E. (1986). Delocalized plans and program comprehension. *IEEE Software*, 3(3), 41–49. <https://doi.org/10.1109/MS.1986.233414>
- Liamputtong, P. (2009). Qualitative data analysis: Conceptual and practical considerations. *Health Promotion Journal of Australia*, 20(2), 133–139. <https://doi.org/10.1071/HE09133>
- Lister, R., Simon, B., Thompson, E., Whalley, J. & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *SIGCSE Bulletin*, 38(3), 118–112. <https://doi.org/10.1145/1140124.1140157>
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Möstrom, J. E., Sanders, K., Seppälä, O., Simons, B. & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4), 119–150. <https://doi.org/10.1145/1041624.1041673>
- Littman, D., Pinto, J., Letovsky, S. & Soloway, E. (1987). Mental models and software maintenance. *Journal of Systems and Software*, 7(4), 341–355. [https://doi.org/10.1016/0164-1212\(87\)90033-1](https://doi.org/10.1016/0164-1212(87)90033-1)

- Loksa, D., Ko, A., Jernigan, W., Oleson, A., Mendez, C. & Burnett, M. (2016). Programming, problem solving, and self-awareness: Effects of explicit guidance. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 1449–1461. <https://doi.org/10.1145/2858036.2858252>
- McCartney, R., Boustedt, J., Eckerdal, A., Sanders, K. & Zander, C. (2013). Can first-year students program yet? A study revisited. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, 91–98. <https://doi.org/10.1145/2493394.2493412>
- Middendorf, J. & Shopkow, L. (2018). *Overcoming student learning bottlenecks: Decode your disciplinary critical thinking*. Stylus Publishing.
- Middendorf, J. & Pace, D. (2004). Decoding the disciplines: A model for helping students learn disciplinary ways of thinking. *New Directions for Teaching and Learning*, 98, 1–12. <https://doi.org/10.1002/tl.142>
- Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2), 81–97. <https://doi.org/10.1037/h0043158>
- Moore, D., Zabrucký, K. & Commander, N. (1997). Validation of the metacomprehension scale. *Contemporary Educational Psychology*, 22(4), 457–471. <https://doi.org/10.1006/ceps.1997.0946>
- Nathan, M. & Petrosino, A. (2003). Expert blind spot among preservice teachers. *American Educational Research Journal*, 40(4), 905–928. <https://doi.org/10.3102/00028312040004905>
- Newen, A. (2015). What are cognitive processes: An example-based approach. *Synthese*, 194, 4251–4268. <https://doi.org/10.1007/s11229-015-0812-3>
- Orlov, P. & Bednarik, R. (2017). The role of extrafoveal vision in source code comprehension. *Perception*, 46(5), 541–565. <https://doi.org/10.1177/0301006616675629>
- Orlov, P., Bednarik, R. & Orlova, L. (2016). Programmers' experiences with working in the restricted-view mode as indications of parafoveal processing differences. *PPIG*, 96–105.
- Oyebode, F. (2018). *Sims' symptoms in the mind: Textbook of descriptive psychopathology* (6th ed.). Elsevier.
- Pace, D. (2017). *The decoding the disciplines paradigm: Seven steps to increased student learning*. Indiana University Press.
- Parcell, E. & Rafferty, K. (2017). Interviews, recording and transcribing. In M. Allen (Ed.), *The SAGE Encyclopedia of Communication Research Methods*. Sage Publications. <http://dx.doi.org/10.4135/9781483381411.n275>
- Patton, M. Q. (2015). *Qualitative research and evaluation methods: Integrating theory and practice* (4th ed.). Sage Publications.
- Pennington, N. (1987). Comprehension strategies in programming. In G. M. Olson, S. Sheppard & E. Soloway (Eds.), *Empirical studies of programmers: Second workshop*. Ablex Publishing Corporation.

- Pero, S., Incoccia, C., Caracciolo, B., Zocolotti, P. & Formisano, R. (2006). Rehabilitation of attention in two patients with traumatic brain injury by means of 'attention process training'. *Brain Injury*, 20(11), 1207–1219. <https://doi.org/10.1080/02699050600983271>
- Plowright, D. (2011). *Using mixed methods: Frameworks for an integrated methodology*. Sage Publications.
- Powell, N., Moore, D., Gray, J., Finlay, J. & Reaney, J. (2004). Dyslexia and learning computer programming. *ACM SIGCSE Bulletin*, 36(3), 242. <https://doi.org/10.1145/1026487.1008072>
- Praveen, A. (2016). Program comprehension and analysis. *International Journal of Engineering and Applied Computer Science*, 1(1), 17–21. <https://doi.org/10.24032/ijeacs/0101/04>
- Preece, J., Rogers, Y. & Sharp, H. (2015). *Interaction design: Beyond human-computer interaction* (4th ed.). John Wiley & Sons.
- Saldaña, J. (2016). *The coding manual for qualitative researchers*. Sage Publications.
- Scalabrino, S., Linaes-Vásquez, M., Poshyvanyk, D. & Oliveto, R. (2016). Improving code readability models with textual features. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 1–10. <https://doi.org/10.1109/ICPC.2016.7503707>
- Shopkow, L., Diaz, A., Middendorf, J. & Pace, D. (2013). From bottlenecks to epistemology in history: Changing the conversation about the teaching of history in colleges and universities. In R. Thompson (Ed.), *Changing the conversation about higher education* (pp. 15–38). Rowman & Littlefield.
- Sohlberg, M. (2000). Psychotherapy approaches. In S. Raskin & C. Mateer (Eds.), *Neuropsychological management of mild traumatic brain injury* (pp. 137–156). Oxford University Press.
- Uwano, H., Nakamura, M., Monden, A. & Matsumoto, K. (2006). Analyzing individual performance of source code review using reviewers' eye movement. *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, 133–140. <https://doi.org/10.1145/1117309.1117357>
- Uzonwanne, F. (2016). Rational model of decision making. In A. Farazmand (Ed.), *Global encyclopedia of public administration, public policy, and governance*. Springer International. https://doi.org/10.1007/978-3-319-31816-5_2474-1
- Van Someren, M., Barnard, Y. & Sandberg, J. (1994). *The think aloud method: A practical guide to modelling cognitive processes*. Academic Press.
- Von Mayrhauser, A. & Vans, A. M. (1995). Program understanding: Models and experiments. *Advances in Computers*, 40, 1–38. [https://doi.org/10.1016/S0065-2458\(08\)60543-4](https://doi.org/10.1016/S0065-2458(08)60543-4)
- Wiedenbeck, S. (1985). Novice/expert differences in programming skills. *International Journal of Man-Machine Studies*, 23(4), 383–390. [https://doi.org/10.1016/S0020-7373\(85\)80041-9](https://doi.org/10.1016/S0020-7373(85)80041-9)
- Wood, D., Brunner, J. S. & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychiatry and Psychology*, 17(2), 89–100. <https://doi.org/10.1111/j.1469-7610.1976.tb00381.x>

Xie, B., Nelson, G. & Ko, A. (2018). An explicit strategy to scaffold novice program tracing. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 344–349. <https://doi.org/10.1145/3159450.3159527>