Thinking in programming

Errol Thompson
E.L.Thompson@massey.ac.nz

Abstract

From the basis of the characteristics of expertise and the ways that students approach programming tasks, this paper explores the thinking patterns or cognitive skills used in programming Finally the paper considers issues of teaching and assessment.

Keywords: metacognition, cognition, patterns.

Introduction

Research on the nature of expertise says that "experts notice features and meaningful patterns of information that are not noticed by novices" (Bransford, et al. 2000). Soloway, et al. (1988) found that expert programmers use programming plans as a way of understanding a program and what it does. Adelson (1981; 1984) describes expert programmers as conceptualising a program in terms of what the program needs to do and a novice conceptualises a program in terms of how the program operates. Do students see the development of these characteristics as important in their approach to class exercises and assessments?

An example

In a programming class using object-oriented techniques, the students have been set an exercise to implement a doubly linked list using a test driven approach. In a lecture session, the lecturer has gone over how a doubly linked list should work. This included drawings of nodes and the changes, especially in the reference pointers, that occur when a node is added or removed. The lecturer has also identified the key methods that would be required and the parameters for those methods. Pseudocode was used to illustrate the logic.

Straight into coding

One student immediately sets in to coding the methods and argues that until he has a working list, it will not be possible to work out how to test it. The lecturer asks how he will know that he has a working list. The response is that he will implement a form to display the current values in the list. But what will be on the form and what will the values on this form really show? The student is unsure but feels that he couldn't test a program without a screen form.

This student quickly starts to code attempting to write the code for all the methods before running any tests. When he switches to testing, the code throws errors on everything the student tries. After some time, he asks for assistance and again the lecturer asks what the student expects out from the test. The student doesn't know. The lecturer examines the code but it doesn't resemble the psuedocode, is poorly structured and very difficult to follow.

The lecturer asks some questions to try and draw out the student's understanding. The student struggles to answer and the lecturer often has to phrase the question so that there should be a simple yes or no response to what a segment of code is supposed to do. The code slowly progresses to a point where it will run and produce an on screen result that looks something like what was expected. The student's tests are minimal and on examination the code fails to reach the intended outcome.

When questioned, the student still has minimal understanding of linked lists and is unable to explain what the code does. The code is full of segments that do nothing and there are large segments of code commented out. The lecturer sees that the code that works are those that have been suggested and there is very little of the student's own code that actually works. The lecturer wonders what would be required for this student to understand.

Wants code to copy

Another student complains that he needs example code in the language that the class is using. Without such an example, he claims that he will not be able to see how a linked list is implemented. The lecturer says that it is important that he should be able to develop a linked list from the information that the class has been given and/or to be able to use examples from other programming languages. However, the student goes away and spends time searching for example code. At the next class, the student says that they cannot do the exercise and that example code is required.

The lecturer goes over the working of a linked list again with this student. With each part of the explanation, the lecturer asks how the student might code this segment. Slowly, the student has most of the code needed to implement the linked list. Some adjustments will still need to be made but the student has actually produced the bulk of the code himself.

At the end of the process, the student is still adamant that they should have been given a more explicit example that they could follow. As well, the student still has minimal ability to explain the workings of a linked list and feels that being able to do so is not relevant to his ability to be able to program.

In another exercise, this student is given example code that is close to what is required but needs amendment. For this second exercise, the student copies the example code changing names of variables and making some of the required amendments. The new code is clearly a plagiarism of the example code. In discussion with the lecturer, the student shows that he has no understanding of what the code does or how the code can be used in other situations.

Working from concepts

A third student reviews the concepts for a linked list and works through some examples. She then looks at each method to be implemented for the linked list and defines the input requirements and then the expected result in the linked list structure. This student then writes a series of tests that will verify that the various attributes of the impacted nodes will have the values expected after the method has executed. The student not only uses the lecturer's drawings and explanations but constructs a number of her own diagrams. She verifies that her tests not only verify the expected changes but that they also test that other attributes are not changed by the method to be tested.

She also realises that the methods will need to be implemented in an appropriate sequence and that there will be differences in the addition of the first value to the list and removal of the last. Having understood how the linked list will work, she then starts to code using her test sequence to drive her coding. On completion of the coding, she is confident that her list works as it is supposed to and can discuss both the working of the code and the operations of a linked list.

Examination of the code shows that the code is clean and precise. The code might be described as a simple solution to implementing a linked list. It is easily understood even though there are minimal comments about the way that it works. The tests also demonstrate a clear understanding of what the code must do in order to work successfully.

Discussion

All three students believe that they have achieved the required outcomes of the exercise. However, only the third student can discuss her code and the way that linked lists work. Her approach to the programming task also showed a willingness to focus on what the program needed to do rather than how it would work. Once she knew how it would work, the writing of the code became easier.

The first student focussed on how the program would operate but lacking the understanding of how it would work, he has code that seems to run without obvious errors but when a testing strategy is applied to the code, it fails miserably. After the exercise, the student has a limited understanding of what a linked list is supposed to do.

The second student has a linked list that works but still doesn't understand the concept. If he is asked to implement a linked list again, he will do so using the code he now has as an example. Will this student overtime develop a conceptual understanding of a linked list? Will he later on be able to implement code for a concept that he hasn't seen an example for?

Which of these students is the most likely to develop the characteristics of an expert programmer?

Other Information System subjects

Is this problem only relevant to programming? Examples could be drawn from papers on systems analysis and design. The students might have some understanding of the tools and techniques but the jump to being able to apply these to an analysis or design exercise leaves them struggling.

Thinking patterns

In endeavouring to help students, a consistent problem is lack of ways of thinking about information systems and their implementation. Possible thinking frameworks such as systems theory are seen as things to learn and not as frameworks for understanding, interpreting and developing Information System solutions.

In terms of programming some of the possible thinking frameworks are:

1. logic patterns,
2. systems thinking patterns,
3. higher-order processing patterns,
4. design patterns, and

5. process or organisational patterns.

Logic patterns

In this category the patterns are those of sequence, conditional, loop, and procedure call. For programming in procedural oriented languages, these are the core foundation. To some extent these become the implicit patterns used by programmers in the writing of all code. Programmers not only understand these patterns but they understand the implications and dependencies in using these patterns.

System thinking patterns

Programmers might readily identify with the input-process-output pattern as being the core foundation of all their programming code. Again though, it may not be explicitly stated. When pushed, they may also identify the use of a hierarchy of systems. The concepts of encapsulation and cohesion rely on the idea of a hierarchy of systems. In recent papers, I have been endeavouring to encourage students to see objects as a system with some success in improving their thinking about object-oriented programming.

Higher-order processing patterns

Where logic patterns provide an initial framework, higher-order patterns such as initialise-process-terminate can provide a higher-level structural pattern to the overall code. Emphasis on the different paradigms used in implementing systems such as procedural/sequential, object-oriented, service-driven, or event-driven can further expand the student's ability to work with complex program structures and logic.

Design patterns

Design patterns can be seen as how techniques. However, they also provide an initial level what focus that implies or drives a how. It is there use as ways of thinking about what a program is doing that I believe may be more beneficial initially in a programming context. An experienced analyst on hearing the client talk of an invoice or order immediately has a pattern in mind of how that would be implemented. The creational, structural, and behavioural design patterns can provide the same thinking tools for the programmer both to writing their own code and interpreting the increasing complex frameworks of programming environments.

Process or organisational patterns

Knowing how to approach a programming task can be a key element in providing the initial ability to accept the challenge of a complex and unfamiliar programming task. Process and/or organisational patterns provide ways of thinking about how to approach different challenges and tasks.

How to teach

When looking at the teaching of programming topics, we need to consider how we can foster the development of these thinking processes. The process as content movement (Costa and Liebmann 1996) would argue that teaching the thinking processes should be central to the educational system. Science is seen as a process of inquiry (Young, 1996) with the currently accepted scientific facts being those that through the process of inquiry have the best supporting evidence.

From a software development perspective, the software development process can be seen as a process of knowledge acquisition (Armour, 2000). The programmer is endeavouring to capture the business knowledge of data and processes and implement in an executable program.

To achieve this, the students need to see the conceptual frameworks as not simply something to be learnt but as ways of thinking about the problem situations that are to be resolved. They also need to see how the thinking patterns can be used both to evaluate their work and to act as a guide for implementing solutions.

The tools and techniques of software development need to be seen as ways of expressing their thinking and understanding of a possible solution. An object model is the expression of the designer's current understanding of the system. It isn't a final solution but a step along the way.

If developing a thinking process is a core requirement then how can a thinking process be corrected? Bransford et al (2000), in discussing the core learning principles. talk of the need to "draw out and work with the pre-existing understandings that their students bring with them". In the case of thinking patterns, this means helping the students make explicit their current thinking patterns. The teaching process can then provide evidence to reinforce or challenge those thinking patterns.

The new thinking patterns, that are introduced, need to be seen by the learner as being relevant to the context in which they are to be applied. Expert's knowledge is conditionalised (Bransford et al 2000). That is they know when what knowledge is appropriate for each problem situation. The student needs to develop this same conditioning of knowledge so that they can make appropriate choices during problem solving.

The emphasis here is to have the students evaluate the way that they are thinking about the programming task and to endeavour to have them revise their thinking based on the perceived effectiveness. The ability to assess and self-regulate

your use of cognitive skills and learning strategies is called metacognition. The development of metacognitive skills has to be a core part of the teaching process (Bransford et al., 2000).

The final aspect is how to assess. The final quality of the code does not reveal the process or the thinking that the student has applied to the task. The assessment must also focus on how the student arrived at the final code. Yet this assessment cannot be based on one best process. It needs to allow for different processes that may all be equally effective.

Conclusion

The emphasis of this paper is the thinking in programming. The underlying contention is that expertise in programming is dependant on the thinking patterns that experts bring to the task. I have endeavoured to illustrate the issues through examples, to identify some of the possible thinking patterns, and briefly to touch on teaching and assessment.

References

Adelson, B. (1981) Problem solving and the development of abstract categories in programming languages. *Memory & Cognition* **9** 422-433.

Adelson, B. (1984) When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **10** 483-495.

Armour, P.G. (2000) The case for a new business model: Is software a product or a medium? *Communications of the ACM* **43** (8): 19-22.

Bransford, J.D., Brown, A.L. and Cocking, R.R. (2000) *How people learn: brain, mind, experience, and school*, Expanded Edition edn. Washington: National Academy Press.

Costa, A.L. and Liebmann, R.M. (1996) *Envisioning process as content: Toward a renaissance curriculum*, Thousand Oaks, CA: Corwin Press Inc.

Soloway, E., Adelson, B. and Ehrlich, K. (1988) Knowledge and processes in the comprehension of computer programs. In: Chi, M.T.H., Glaser, R. and Farr, M.J., (Eds.) *The nature of expertise*, pp. 129-152. Hillsdate, NJ: Lawrence Erlbaum Associates

Young, D.B. (1996) Science as inquiry: Transforming science education. In: Costa, A.L. and Liebmann, R.M., (Eds.) *Envisioning process as content: Toward a renaissance curriculum*, Thousand Oaks, CA: Corwin Press Inc