National Centre for Computing Education

Raspberry Pi

Understanding program comprehension using the Block Model

In recent years, program comprehension has been recognised as an important step in learning to program. It is a step that is easily missed as learners dive straight into writing programs before they have learnt to read them. What exactly is program comprehension, why is it so important, and how can educators develop these skills with their learners?

Macro structure **(M)**	• Annotate code or draw a diagram to show the overall structure • Restructure an 'untidy' program	• Identify inputs needed to test all program branches • Will line X ever be executed?	• Choose a name for a given program • Select/write a sentence that describes a program's purpose
Relationships **(R)**	• Identify variable scope • Highlight function calls	• Draw the flow of control • Find redundant conditional branches	• Choose a name for a variable/function • Are two programs/segments functionally equivalent?
Blocks **(B)**	• Identify block types, such as finite loops, 'else' conditions, function definitions, etc	• Reordering lines of code • Parson's Problems	• Explain the purpose of a block of code
Atoms **(A)**	• Identify statement types, such as assignments and conditions	• Trace values through a program	• Explain the purpose of a single line
	Text surface **(T)**	Program execution **(P)**	Function/purpose **(F)**

Summary

Learning programming has several challenges:

- It is concept-rich, leading to cognitive overload
- It balances comprehension with coding experience
- It demands persistence and resilience
- Learners need a secure mental model of computation

Program comprehension

- It allows learners to interpret, explain, adapt, debug, and create programs
- It supports learners to develop programming patterns or plans
- It can be divided into 12 'zones' using the Block Model
- Learners should develop knowledge in each zone and be able to move between them

Comprehension tasks

- Educators can use this Block Model to categorise tasks and identify gaps where students need support
- A range of strategies already exist that have been mapped to the Block Model

Why do students find programming challenging?

Although programming is a valuable and rewarding skill to learn, many learners find the process challenging:

- Even simple programs are rich in concepts that can cause cognitive overload in learners
- Learners may rush to write programs too soon, before they have read and understood the relevant concepts
- Programs often don't work first time, requiring resilience and persistence from learners
- Learners need to switch between different abstractions, the problem, the program text, and its execution, constantly moving from single lines to the program as a whole
- Learners also need a mental model or notional machine for how the computer works and will execute the program

These challenges do not mean that programming is intrinsically difficult, and recognition of these challenges can help educators identify where they can support their learners.

Program comprehension

Experienced programmers demonstrate a high degree of program comprehension. As well as having a robust notional machine, they can develop programming 'plans' (chunks of code that perform a specific task), based on common features in programs that they have seen. They can then use these plans or patterns to interpret, explain, adapt, debug, and create programs.

Novice programmers know of very few (if any) programming plans, and have limited awareness of how programs are executed (notional machine). Their focus may be limited to decoding individual words in a program, rather than comprehending their meaning or the meaning of the wider program. As educators we need to understand how to bridge this gap.

Comprehension tasks

There are already many great examples of activities that promote program comprehension, including tracing, Parson's Problems, PRIMM, and tasks in which learners 'explain the purpose'. A teacher-focused study[1] identified more than 60 different activities that could support learners in developing program comprehension skills. It also highlighted that many of these activities were already used, to assess program comprehension, rather than support its development.

As program comprehension is quite broad and there are a number of activities to choose from, it can be difficult for educators to know which activities to use in which circumstances.

The Block Model

A useful tool for understanding and categorising aspects of program comprehension is the Block Model.[2] This framework captures what level the learner is focused on:

- **Atoms,** the smallest element, are the keywords, symbols, and syntax, or a single line of code.
- **Blocks** are small chunks of code that perform a task, eg single lines, loops, selection statements, or functions.
- **Relationships** are the connections between blocks, and the manner in which they work together, such as function calls and return values.
- **Macro structure** refers to the program as a whole.

The framework also considers the 'dimension' of the program, or how the learner is viewing it:

- The program exists as a static piece of text. This is called the **text surface** and is where learners need to consider the grammar and syntax of their program.
- When the program is executed, it becomes a dynamic object that may behave differently depending on its inputs. This dimension is known as the **program execution**.
- The **function** solely concerns the purpose of the program or code snippet.

The Block Model therefore comprises 12 zones of program comprehension that learners should be able to move between as they develop their understanding. The related 'holey quilt' theory[3] suggests that learners begin with varying levels of knowledge in each zone, ranging from fragile to deep. Knowledge is deepened over time and can be supported by learning activities targeting each zone.

Mapping tasks to the Block Model

It is important to devise activities that develop comprehension in each of these 12 zones, in order to support learners' full understanding of the program. By considering each of the three dimensions in turn, we can identify tasks that may foster comprehension at each level of focus.

Comprehending the text surface can be tricky, as learners need to discern the meaning from text with unfamiliar terms, structures, and syntax. Without support, they may get stuck focusing on the program at the 'Atoms' level. A simple strategy that works at all levels of focus is to identify aspects of the code within the text. By identifying examples of variables, conditions, finite loops, functions, etc, educators can help learners make sense of the text, and connect it to underlying concepts.

During program execution, several approaches could be used to help learners develop their understanding and their mental models. For example, learners could trace simple programs, determining the end state of variables or the inputs required to reach a specific state. Learners could also complete Parson's Problems, which transcend the text surface and enable learners to focus on the correct sequence of instructions for a specified goal. Similarly, learners could investigate the effect of swapping two lines of code, or try to find lines that can never run. Note that many of these activities are also good examples of the 'Investigate' phase of the PRIMM methodology.[4]

Learners can also benefit from exploring function. Asking learners to explain the function of a line, snippet, or entire program is a great place to start. To do this, they will have to use clues within the text and observe the execution. Educators can vary the degree of challenge by the clues they leave in the programs. Also, educators can connect function back to text by asking learners to provide meaningful names for variables, functions, or entire programs. Alternatively, learners could be given a description of the purpose and identify a program that matches, or compare multiple programs to find which are functionally equivalent.

There are lots of options for educators to choose from, but the most important step is to review our own practice, to find and fill those gaps in learners' program comprehension.

Was this Pedagogy Quick Read useful? Share your thoughts and feedback at: ncce.io/pedagogyfeedback

References

1 Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., Mirolo, C. & Weeda, R. (2019) Program comprehension: Identifying learning trajectories for novice programmers. In: *ITiCSE '19: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. New York, ACM. pp. 261–262.

2 Schulte, C., Clear, T., Taherkhani, A., Busjahn, T. & Paterson, J. H. (2010) An introduction to program comprehension for computer science educators. In: Clear, A. & Russell Dag, L. (eds.) *ITiCSE-WGR '10: Proceedings of the 2010 ITiCSE working group reports*. New York, ACM. pp. 65–86.

3 Clear, T. (2012) The hermeneutics of program comprehension: a 'holey quilt' theory. *ACM Inroads*. 3(2), 6–7.

4 Sentance, S. (2020) The I in PRIMM. *Hello World*. 14, 50–53.

Raspberry Pi