Teaching Programming: Insights from Research for Classroom Practice

Violetta Lonati

University of Milan - Department of Computer Science

23 April 2026, Lausanne

Teaching programming

Programming is the essence of computing/informatics. Indeed, computing is much more than programming, but programming is essential to computing.

[Caspersen (2018). *Teaching Programming. In Computer Science Education*, Bloomsbury Academic]

Ideas from the research

- Syntactical, conceptual, strategic knowledge
- The notional machine
  - Misconceptions
  - Tracing
- Program writing Vs program comprehension
  - BRACElet project
  - EIPE - Explain in Plain English
  - Program comprehension tasks
  - The Block Model
- Strategical knowledge
  - Variables' roles
  - Soloway's goals & plans
  - Composition of plans
- Learning to program in the age of Generative AI
  - Automatic programming
  - GenAI in software development and in programming learning

Outline

Syntactic, conceptual, strategic knowledge

Different types of knowledge and skills have to be built when learning to program:

- **syntactic knowledge**, i.e., knowledge of the language features
- **conceptual knowledge**, i.e., knowledge of the semantics of the various constructs (notional machine)
- **strategic knowledge**, i.e. the ability to use syntactic and conceptual knowledge in the most appropriate and effective way to solve programming problems. Strategic knowledge also includes the ability to exploit and combine solutions to known problems to solve new ones.

Example

```c
int main( void ) {
  int n, x = 0;

  do {
    scanf( "%d", &n );
    x = x + n;
  } while ( n != 0 );

  printf( "%d\n", x );
  return 0;

}
```

Syntactic knowledge

syntax of the `do-while` loop or of the += operator in C

Conceptual knowledge

how many times the loop is executed, when it stops

Strategic knowledge

use of variable x as an *accumulator* to compute the sum of the read values

Outline

The notional machine

> *The notional machine is an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed.*

[du Boulay, B. (1989). *Some difficulties of learning to program*.]

The notional machine

> *The notional machine is an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed.*

[du Boulay, B. (1989). *Some difficulties of learning to program*.]

The purpose of make the notional machine explicit is to give a basis to understand the beahavior of programs in execution.

One abstraction level “below”

- A program written in a programming language is executed by a machine.
- The machine “understands?” (i.e., can mechanically execute) the instructions of that language.
  - “Scratch machine”, “Python machine”

One abstraction level “below”

- A program written in a programming language is executed by a machine.
- The machine “understands?” (i.e., can mechanically execute) the instructions of that language.
  - “Scratch machine”, “Python machine”
- Not physical machines, but abstract machines

One abstraction level “below”

- A program written in a programming language is executed by a machine.
- The machine “understands?” (i.e., can mechanically execute) the instructions of that language.
  - “Scratch machine”, “Python machine”
- Not physical machines, but abstract machines

“As the level of abstraction in computing education across educational levels steadily arose, the credo among computing cognoscenti became that one needs to be familiar with at least one abstraction level below that at which one is working.”

[Tedre et al. (2021). *Teaching Machine Learning in K-12 Classroom*.]

Misconceptions

The abstract model of the machine in novices is incomplete or inaccurate and this is often source of misconceptions.

Misconceptions

The abstract model of the machine in novices is incomplete or inaccurate and this is often source of misconceptions.

> *In some disciplines, concepts and phenomena are largely negotiable and up for interpretation. [...] However, computing also features many concepts that are precisely defined and implemented within technical systems. [...]*
>
> *Sometimes a novice programmer "doesn't get" a concept or "gets it wrong" in a way that is not a harmless (or desirable) alternative interpretation.*
>
> *Incorrect and incomplete understandings of programming concepts result in unproductive programming behavior and dysfunctional programs.*

[Sorva, J. 2013. *Notional machines and introductory programming education*. ACM Transactions on Computing Education]

Misconceptions

Definition of “misconception” in programming

The term misconception refers to any understanding that is deficient or inadequate for many practical programming contexts.

Other terms: “partial understandings”, “incorrect understandings”, “student-constructed rules”, “mistakes”, “errors”.

Misconception catalogue Appendix A in Sorva’s PhD Thesis.

Program tracing

- Tracing is defined as following the execution of a program, line by line, following the control path.
- While tracing, we must keep track of our current position in the program and also the state of the variables after each line is executed, possibly using a tracing table.

T1: What do the variables v1, v2 and v3 hold after the following Python code is executed? Assume that they are all integer type variables .

```
v1 = 10;
v2 = 15;
v3 = v1;
v1 = v2;
v2 = v3;
```

T2: What is the output of the following code segment?

```
int count=0;
for (int i=0; i<N; i++) {
  if (i % 2 == 0) {
    count++;
}}
System.out.println(count);
```

T3: What is the output of the following code segment?

```
int i, j;
for (i = 1; i<=5; i++) {
  for (j = 1; j<=5; j++) {
    printf("%3d", i*j);
}}
printf("\n");
```

Program tracing

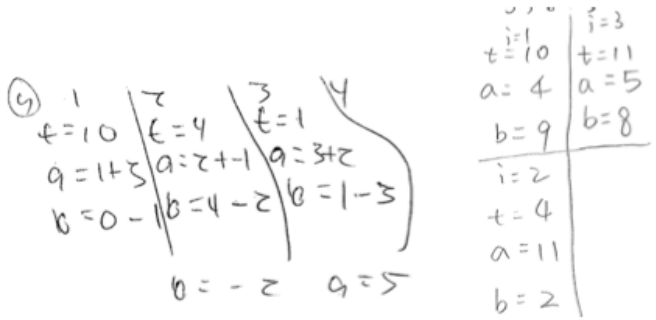


Figure 3: Examples of the "chunk" technique

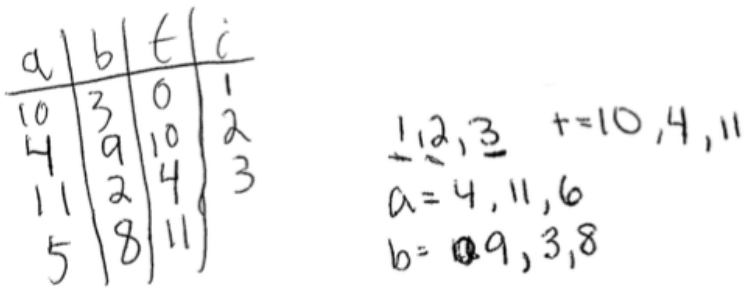


Figure 4: Examples of the "line" technique

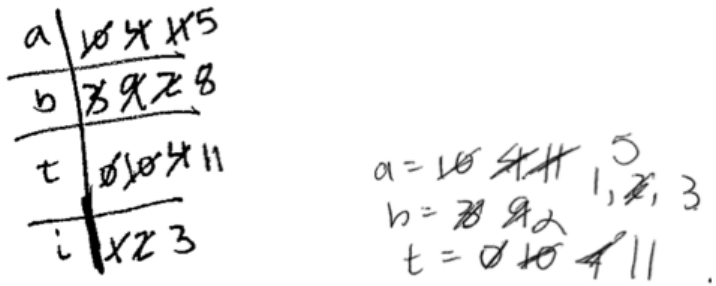


Figure 5: Examples of the "crossout" technique

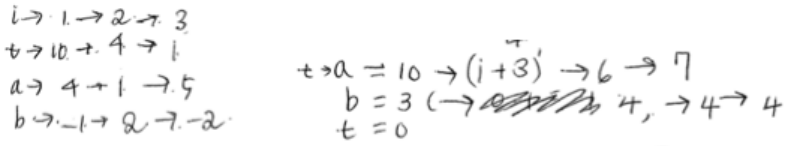


Figure 6: Examples of the "arrow flow" technique

[Cunningham et al. (2017) *Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw*. ICER 2017]

Purposes of tracing

Students do not realize that tracing is a skill they need to acquire in order to learn programming.

Purposes of tracing

Students do not realize that tracing is a skill they need to acquire in order to learn programming.

Tracing:

1. is a method the instructor uses to show how the constructs work.
2. is a programming strategy when searching for a bug.
3. is not an end but a source for the real task of understanding code
4. is an opportunity to build understanding of the program execution (notional machine)
5. is a tool to diagnose students' misconceptions

Outline

Writing skills Vs program comprehension

- Programming courses are often centered on code writing.
- It is typically expected by instructors that the program comprehension skill will be learned as a side-effect of learning to write programs.

Writing skills Vs program comprehension

- Programming courses are often centered on code writing.
- It is typically expected by instructors that the program comprehension skill will be learned as a side-effect of learning to write programs.
- Actually, students often lack even basic tracing ability, i.e. executing pieces of code step by step simulating the machine.

Writing skills Vs program comprehension

- Programming courses are often centered on code writing.
- It is typically expected by instructors that the program comprehension skill will be learned as a side-effect of learning to write programs.
- Actually, students often lack even basic tracing ability, i.e. executing pieces of code step by step simulating the machine.
- There is strong correlation between novices' tracing, reading and writing skills.

[Clear, T. et al. (2011). *Report on the final BRACElet workshop*]

Writing skills Vs program comprehension

Program comprehension, tracing and writing skills are highly correlated.

> *We believe that the more likely explanation for a correlation between writing code and explaining code is that both writing and explaining depend upon a common set of skills concerned with reasoning about programs.*
> *If reasoning about code is the underlying skill that is common to both code writing and code explaining, then the crucial pedagogical question is how to most efficiently develop that underlying skill.*

[Lister et al.(2014). *'Explain in plain english' questions revisited: data structures problems*. SIGCSE '14. ACM]

EiPE - Explain in Plain English

- Novices tend not to abstract beyond the concrete code, they have difficulties at summarizing the purpose of a piece of code.
- Use of SOLO taxonomy to analyse answer to EiPE tasks

SOLO category	Description
Relational [R]	Provides a summary of what the code does in terms of the code's purpose.
Multistructural [M]	A line by line description is provided of all the code. Summarisation of individual statements may be included
Unistructural [U]	Provides a description for one portion of the code (i.e. describes the if statement)
Prestructural [P]	Substantially lacks knowledge of programming constructs or is unrelated to the question
Blank	Question not answered

[Whalley J., Lister R, et al. (2006). *An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies.*] and follow-up papers

How to teach program comprehension skills?

How to teach program comprehension skills?

- Program comprehension, *i.e.*, the process of building a "mental model of a program" is an important skill to acquire in order to develop good programming skills.
- A mix of code-writing and code-reading is deemed to be the most effective way for students to ultimately develop good programming skills

Program Comprehension (*ProgComp*) – usually conceptualized as a process in which an individual constructs his or her *mental model* of a program.

***ProgComp* task** – a task through which the learner encounters an artifact that represents the program. The task asks the learner to engage with the artifact in some way. Through this interaction with the artifact, the learner is stimulated to build, elaborate on and refine their mental model.

The artifact is typically source code, but might also be another form of specification such as the nodes in Parsons problems [82] or a collection of blocks in a block-based programming language such as Scratch [66].

[Izu et al. (2019) *Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories*. WG at ITiCSE 2019]

The Block Model

(M) Macro structure	Understanding the overall structure of the program text	Understanding the *algorithm* underlying a program	Understanding the goal/purpose of the program (in the context at hand)
(R) Relationships	Relations & references between blocks (e.g. method calls, object creation, data access...)	Sequence of method calls, *object sequence diagrams*.	Understanding how subgoals are related to goals, how function is achieved by subfunctions
(B) Blocks (Chunks)	*Regions of Interest* (ROI) that syntactically or semantically build a unit	Operations of a block, a method, or a ROI (chunk from a set of statements)	Understanding the function of a block, seen as a subgoal
(A) Atoms	Language elements	Operation of a statement	Function of a statement: its purpose can only be understood in a context
	(T) Text Surface	**(P) Program Execution**	**(F) Function/Purpose**
Duality	Architecture/Structure Dimensions		Relevance/Intention Dimension

[Schulte (2008). *Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching*. ACM ICER '08]

Outline

Syntactic, conceptual, strategic knowledge

Different types of knowledge and skills have to be built when learning to program:

- **syntactic knowledge**, i.e., knowledge of the language features
- **conceptual knowledge**, i.e., knowledge of the semantics of the various constructs (notional machine)
- **strategic knowledge**, i.e. the ability to use syntactic and conceptual knowledge in the most appropriate and effective way to solve programming problems. Strategic knowledge also includes the ability to exploit and combine solutions to known problems to solve new ones.

Strategic knowledge is one of the hardest skills to acquire when learning to program.

Variables' roles

Almost 99% of variables encountered in CS1 practice are covered by 8 roles:

- Fixed value
- Stepper
- Follower
- Most recent holder
- Most wanted holder
- Gatherer
- One way flag
- Temporary

[P. Byckling and J. Sajaniemi (2006), *Roles of variables and programming skills improvement*]

Variables' roles (2)

Ruolo	Plan	Esempi	Definizione informale
Accumulatore (*Gatherer*)	Calcolo di un totale	`sum`, `total`, `new_string`	Accumula i contributi di valori individuali
Passo-passo (*Stepper*)	Conteggio / posizione nella Ricerca lineare	`count`, `index`	Assume una successione di valori che variano in modo sistematico e predicibile
One-way flag	Ricerca lineare	`found`, `errorsOccurred`	Una variabile a due valori (spesso booleana) che può cambiare valore una volta sola
Ricercato (*Most-wanted holder*)	Ricerca del valore estremo	`max`, `min`	Contiene il miglior risultato ottenuto finora
Inseguitore (*Follower*) e inseguito (detti anche osso e segugio)	Elaborazione su valori adiacenti	`previousValue` e `currentValue`	Variabile (l'inseguitore) che viene sempre aggiornata con il vecchio valore di un'altra variabile (l'inseguito)

Strategic knowledge: goals and plans

Differences between experts and novices:

- Novices read code line-by-line, experts look for "beacons" and use them to identify "chunks" (meaningful pieces of code).
- Knowledge and use of schema/plan/pattern, both in reading and in writing.

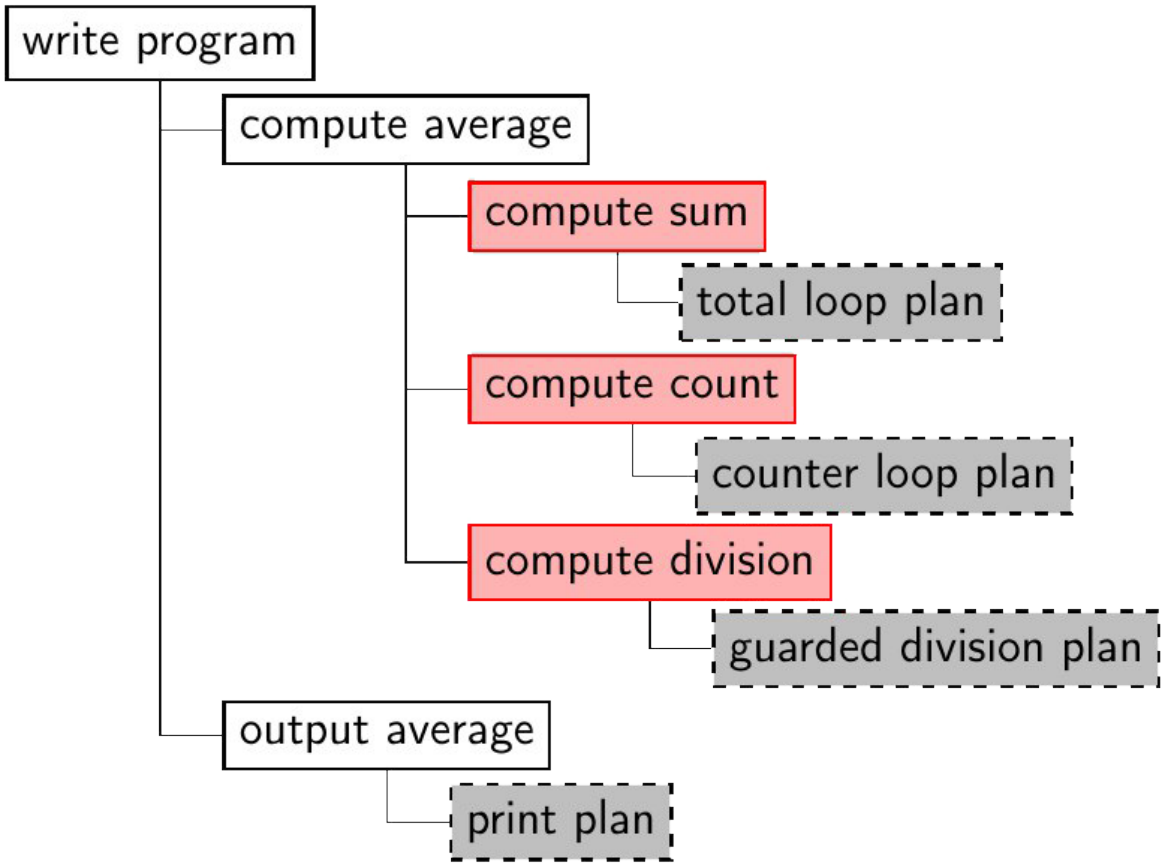
Soloway's terminology

- Goals = basic tasks that need addressing
- Plans = stereotypical canned solutions for typical goals

[E. Soloway (1986). *Learning to Program = Learning to Construct Mechanisms and Explanations*]

Strategic knowledge: goals and plans

Example: compute the average of a sequence of numbers



Examples of plans

Loops for processing items in a collection

- Searching loops
  - Linear Search
  - Guarded Linear Search
- Processing all the items in a collection
  - Process All Items
  - One Loop for Linear Structures
  - Extreme Values

`http://users.cs.duke.edu/~ola/patterns/plopd/loops.html`

Composition of plans

- Abutment: the plans are written one after the other.
- Nesting: plans are written one nested into the other.
- Merging or interleaving: plans are merged together, they need to be deconstructed and reconstructed.
- Tailoring: the plan need adaptation to the problem.

[E. Soloway (1986). *Learning to Program = Learning to Construct Mechanisms and Explanations*]

Students struggles especially with plan merging/interleaving.

[Ginat et al. (2013). *Novice difficulties with interleaved pattern composition.*]

Outline

Learning to program in the age of Generative AI

Prather et al. (2023) *The Robots Are Here: Navigating the Generative AI Revolution in Computing Education.* ITiCSE WG report

Learning to program in the age of Generative AI

Prather et al. (2023) *The Robots Are Here: Navigating the Generative AI Revolution in Computing Education.* ITiCSE WG report

. . . and many, many, many more. . .

Automatic programming

"One of the goals (or dreams) throughout the history of programming language technology, has been "automatic programming" – the ability to automatically generate computer code starting from a high(er)-level description of the specification of that code"

[Simone Martini (2024). *Teaching Programming in the Age of Generative AI.* ITiCSE 2024]

In the ’50s - automatic coding

“[the] function [. . .] of the coder, time-consuming and fraught with mistakes”.
“Automatic coding [. . .] can release the coder from most of the routine and drudgery of producing the instruction code.”
“the replacement of the coder by the computer.”

[Grace Hopper (1954-1955). *Automatic programming – Definitions + Automatic coding for digital computers*]

In the ’50s - automatic coding

“[the] function [. . .] of the coder, time-consuming and fraught with mistakes”.
“Automatic coding [. . .] can release the coder from most of the routine and drudgery of producing the instruction code.”
“the replacement of the coder by the computer.”

[Grace Hopper (1954-1955). *Automatic programming – Definitions + Automatic coding for digital computers*]

Cited by [Bullynck and De Mol (2024) *The Myth of the Coder. Considering the historical distinction between coder and programmer.* CACM]

In the ’50s - automatic coding

“La codification se fait pendant deux phases consécutives:

- le programme est placé sur l’entrée de la machine; la calculatrice exécute une série de calculs qui fournissent le même programme, mais enregistré sous forme d’instructions codifiées, propres à être interprétées automatiquement;
- la calculatrice réalise les calculs, d’après les instructions codifiées.”

[Böhm (1952), *Calculatrices digitales*. PhD thesis.]

In the ’50s - automatic coding

The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†,
H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†,
H. STERN†, I. ZILLER†, R. A. HUGHES§, AND R. NUTT‖

[1957]

In the ’80s – automatic programming

“We will be saying much the same thing about *automatic programming* in 1998 that we said in 1958: that it has improved programmer productivity dramatically and has further reduced the distinction between programmers and end users.”

[Rich and Waters (1988), Automatic Programming: Myths and Prospects. 1988]

In the ’80s – automatic programming

“We will be saying much the same thing about *automatic programming* in 1998 that we said in 1958: that it has improved programmer productivity dramatically and has further reduced the distinction between programmers and end users.”

[Rich and Waters (1988), Automatic Programming: Myths and Prospects. 1988]

“Automatic programming always has been a euphemism for programming with a higher-level language than was then available to the programmer. Research in automatic programming is simply research in the implementation of higher-level programming languages.‘”

[Parnas (1985). *Software aspects of strategic defence systems*. CACM]

Generative AI in Software Development

Generative AI tools can support:

- Code writing and completion
  It can generate functions, suggest code completions, and even translate code between programming languages, speeding up development and reducing repetitive tasks.
- Debugging and error analysis
  It helps identify bugs, explain error messages, and suggest possible fixes, making troubleshooting faster and more efficient.
- Explain complex code

[ChatGPT, free version - yesterday]

Learning to program in the age of Generative AI

- Shift in skill focus
  Learning to program increasingly includes knowing how to prompt AI effectively, review its output, and debug or adapt generated code – program comprehension skills.

Learning to program in the age of Generative AI

- Shift in skill focus
  Learning to program increasingly includes knowing how to prompt AI effectively, review its output, and debug or adapt generated code – program comprehension skills.
- Greater emphasis on fundamentals
  fundamentals become more important, because they help you judge whether AI-generated solutions are correct and efficient.

[Adapted from ChatGPT, free version - yesterday]

Learning to program in the age of Generative AI

The aims of introductory courses are students' development of notional machines for reasoning about how a computer executes a program, and the development of the pragmatic skills for writing and debugging programs that computers can execute.

[Shapiro et al., How Machine Learning Impacts the Undergraduate Computing Curriculum. CACM 61(11), 2018]

Learning to program in the age of Generative AI

The aims of introductory courses are students' development of notional machines for reasoning about how a computer executes a program, and the development of the pragmatic skills for writing and debugging programs that computers can execute.

[Shapiro et al., How Machine Learning Impacts the Undergraduate Computing Curriculum. CACM 61(11), 2018]

Students needs to learn about a new kind of notional – that of the statistical model. The next programming language might be English, prompts may become the new programs, with a non-deterministic machine underneath.

[Lodi and Martini (2025). *Le Notional Machine per l'apprendimento della programmazione, anche nell'era dell'AI generativa*]

Conclusions

- Syntactical, conceptual, strategic knowledge
- The notional machine
  - Misconceptions
  - Tracing
- Program writing Vs program comprehension
  - BRACElet project
  - EIPE - Explain in Plain English
  - Program comprehension tasks
  - The Block Model
- Strategical knowledge
  - Variables' roles
  - Soloway's goals & plans
  - Composition of plans
- Learning to program in the age of Generative AI
  - Automatic programming
  - GenAI in software development and in programming learning

Conclusions

Brilliant student succeed in developing:

- a good notional machine
- strategic knowledge, bulding a catalogue of useful programming plans/schema
- the ability to reason about code
- the ability to effectively use generative AI tools to support coding, debugging, and learning

Conclusions

Brilliant student succeed in developing:

- a good notional machine
- strategic knowledge, bulding a catalogue of useful programming plans/schema
- the ability to reason about code
- the ability to effectively use generative AI tools to support coding, debugging, and learning

Other students do struggle.

Conclusions

Brilliant student succeed in developing:

- a good notional machine
- strategic knowledge, bulding a catalogue of useful programming plans/schema
- the ability to reason about code
- the ability to effectively use generative AI tools to support coding, debugging, and learning

Other students do struggle.

But we have good strategies to support their learning process!