UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Learning to program in a constructionist way

D. Malchiodi

Programming

Programming: producing a precise description of how to carry out a task or to solve a problem.

An interpreter, *different from the producer of the description*, can understand it and effectively carry out the task as described.

Why can it be difficult?

Different concerns (not necessarily separated neither in theory nor in practice)

- "Solve problems" (!)
- Describe solutions "computationally"
  - managing the description itself (the text of the code),
  - considering the actions that take place when the program is run by the interpreter
- Collaborate with others

The dichotomy between static visible code and its implicit dynamics emerges as a critical issue when learning to program.

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Learning outcomes

Learners should develop:

- a perception of programming that does not reduce to production of code, but includes *relating instructions to what will happen* when the program is executed, and eventually comes to include producing applications for use and seeing it as a way to solve problems;
- a mental model of a *notional machine* that allows them to make the association (static) syntax - (dynamic) semantics and to trace program execution correctly and coherently;
- a sense of the advantage to formulate “solutions” in a way that allows a processing agent to carry them out (*computational thinking*)
- an appreciation of the intrinsic collaborative nature of modern software artifacts, where everything depends on components designed by others, and has the potential of being used in new contexts.

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Constructionism

Constructionism [Papert and Harel 1991]:

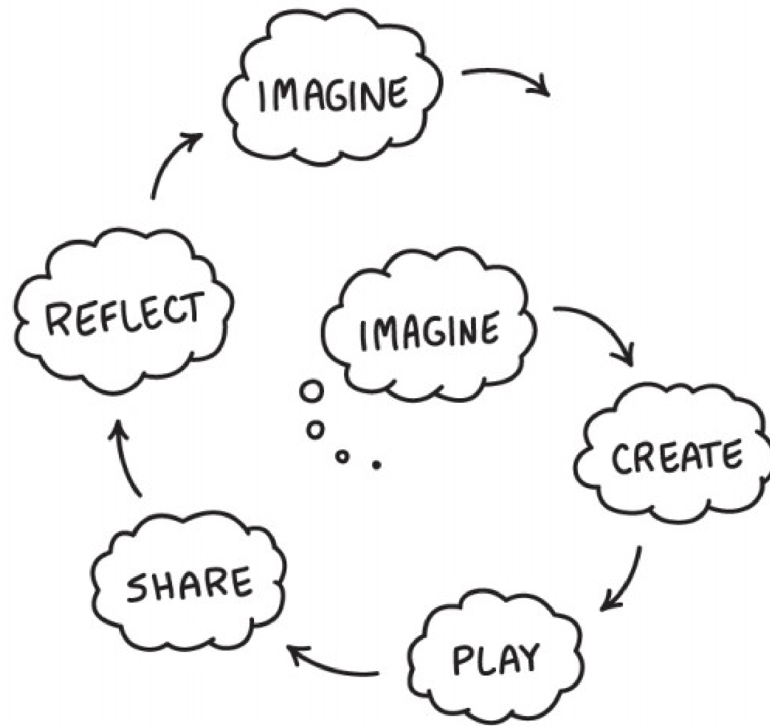
- *strategy* of education which has its roots in Piaget's constructivist theory of learning as an active process;
- people actively construct knowledge *from their personal experience* of the world;
- students do not just receive pre-built ideas from teachers: they have to make them up by engaging themselves with problems, projects, and other people (instructors, but also peers);
- personally-meaningful goals and public artifacts (not necessarily concrete ones: either "a sand castle on the beach or a theory of the universe") that can be shared and discussed with others;
- four P-words: Projects, Peers, Passion, Play.

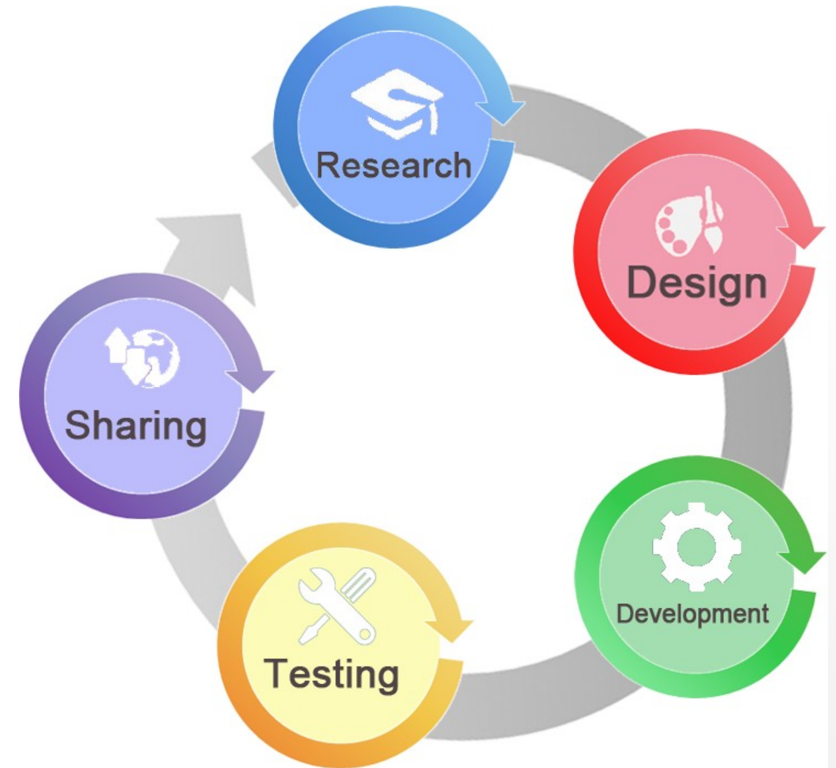
Why constructionism?

- motivation: programming tasks should be engaging and significant to learners;
- a constructivist approach: the construction of knowledge is to be fostered, through activities as much as suitable as possible to *group work and shared meta-cognition*;
- empowering people to do “stuff” is more important than acquiring specific (job-relevant) skills, logical/computational aspects of programming more relevant than “real-world” syntax;
- executable programs are *concrete objects* to talk about and to tinker with.

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Creative learning spiral [Resnick, 2017]





...vs iterative software development

Unplugged approaches

- CSUnplugged!
- Algomotricity: it starts unplugged, but ends with a computer-based phase to close the loop with pupils' previous acquaintance with applications

Unplugged activities can be an important step forward:

- a constructivist environment: tangible objects, dramatised processes, seem to be more suitable to shared meta-cognition.
- suitable in very different (social/economical) contexts;
- no undesired technological hurdles: they do not involve the use of technology, with which not all teachers are at ease;

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Notional machine

When a group of people program the same ‘machine’, a shared semantics is in fact given, but novices do not necessarily write their programs for the formal interpreter they use, rather for the **notional machine** they actually have in their minds:

- an abstract computer responsible for executing programs of a particular kind;
- it normally encompasses an idealized version of the interpreter and other aspects of the development and run-time environment;
- it should bring also a complementary intuition of what cannot be done, at least without specific directions of the programmer.

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Abstract programming patterns

Help novices in recognizing how a relatively low number of abstract patterns can be applied to a potentially infinite spectrum of specific conditions.

- Role of variables: knowledge about use cases of variables (mutable values) in order to solve recurring problems.
- Loop patterns.
- Recursive strategies.

But also, teachers should be familiar and able to recognize common misconceptions, i.e., *understandings that are deficient or inadequate* for many practical programming contexts and learn to ask the right questions.

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Programming environments

- LOGO (1967) «in teaching the computer how to think, children embark on an exploration about how they themselves think» [Papert, 1980]
- Smalltalk (1976) «everyone should be comfortable with programming and computing devices should become ubiquitous in learning environments "along the lines of Montessori and Bruner"»
- BASIC (1964)
- Pascal (1970)
- Lisp, Scheme, Racket

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Visual programming

- Scratch & Co.
- Liveness
- No error messages
- Execution made visible
- Making data concrete
- Open source
- Remixing

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Potential of agile methods

Agile methodologies seem to fit well for constructionist teams of learners:

- typically small groups of 4–8 co-workers;
- agile values: *individuals and interactions* over processes and tools; customer collaboration over contract negotiation; responding to change over following a plan; *working software* over comprehensive documentation;
- self-organizing and emphasis on the need of reflecting regularly on how to become more effective;
- pair programming, test driven development, iterative software development, continuous integration are very attractive for a learning context.

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Work is needed

- Learning to program often lacks deep constructionism. . .
- To set up effective constructionist activities educators and learners should probably be more aware of the so-called notional machines and be more explicit about the complex relationship between the code one writes and the actions that take place when the program is executed.
- Micro-patterns can be exploited in order to enhance problem solving skills of novice programmers, so that they become able to think about the solution of problems in the typical way that make the former suitable to automatic elaboration.
- Agile methodologies are constructivist by nature, with their stress on team working and emphasis on having running artifacts through all the development cycle, constructionist groups of learners can definitely benefit from them.

Sources

M. Monga, M. Lodi, D. Malchiodi, A. Morpurgo e B. Spieler, Learning to Program in a Constructionist Way, in V. Dagiene E. Jasute (Eds.), Constructionism 2018: Computational Thinking and Educational Innovation: conference proceedings, Vilnius University (ISBN 9786099576015), 906–929, 2018, http://dx.doi.org/10.15388/ioi.2019.07

Lodi M., Malchiodi D., Monga M., Morpurgo A. e Spieler B., Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey, Olympiads in Informatics 13 (2019), 99—121

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA