

Spice-Up your Coding Lessons with the ACME Approach

(Discussion Paper)

Giansalvatore Mecca¹, Donatello Santoro¹, Nazzareno Sileno¹ and Enzo Veltri¹

¹Università degli Studi della Basilicata (UNIBAS), Potenza, Italy

Abstract

It is nowadays considered a fundamental skill for students and citizens the capacity of undertaking a problem-solving process in various disciplines (including STEM, i.e. science, technology, engineering and mathematics) using distinctive techniques that are typical of computer science. These abilities are usually called Computational Thinking and at the roots of them stands the knowledge of coding. With the goal of encouraging Computational Thinking in young students, we discuss tools and techniques to support the teaching and the learning of coding in school curricula. It is well known that this problem is complex due to the fact that it requires abstraction capabilities and complex cognitive skills such as procedural and conditional reasoning, planning, and analogical reasoning. In this paper, we present ACME (“Code Animation by Evolved Metaphors”) that stands at the foundation of the Diogene-CT code visualization environment and methodology. We discuss visual metaphors for both procedural and object-oriented programming. Based on them, we introduce a playground architecture to support teaching and learning of the principles of coding. To the best of our knowledge, this is the first scalable code visualization tool using consistent metaphors in the field of Computing Education Research (CER).

Keywords

Coding Tools and Techniques, Learning Environments, Educational Support, Computers and Education

1. Introduction and Motivation

The ability to face problem-solving challenges of STEM disciplines is nowadays universally considered as a crucial skill [1, 2, 3]. *Computational Thinking* has been defined as the capacity of undertaking a problem-solving process in various disciplines using techniques that are distinctive of computer science [4]. At the core of these techniques stand the skill of computer programming, commonly called *coding*. Coding abilities are a fundamental building block of any computational-thinking based approach to teaching. In view of this, this paper illustrates a new method and tools to simplify the teaching and learning of coding. This is a complex problem, that poses several important challenges. The prominent one is that acquiring coding skills requires abstraction capabilities: the conceptualization of the data, the choice of a proper representation, the algorithms development, the semantics of the program (the execution of a program consists of a sequence of actions executed by the machine, and for novice programmers,

SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy
✉ giansalvatore.mecca@unibas.it (G. Mecca); donatello.santoro@unibas.it (D. Santoro);

nazzareno.sileno@gmail.com (N. Sileno); enzo.veltri@unibas.it (E. Veltri)

>ID 0000-0002-1189-1481 (G. Mecca); 0000-0002-5651-8584 (D. Santoro); 0000-0001-9947-8909 (E. Veltri)

 © 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

this is done within a “black box”). Moved by these motivations our ACME approach [5] makes several important contributions towards the goal of fostering the ACME adoption of innovative tools and methods to teach computer programming:

- It introduces a set of consistent metaphors to introduce the concepts of programming. By adopting the metaphors, instructors can work with students through the concepts of programming with a very concrete, visually perceivable counterpart to all of the inner workings of a programming language, in order to ease the approach of students to coding, reduce their cognitive load and the “black box” effect associated with source code. The metaphors consider both introductory coding, i.e., procedural-programming concepts like variables, assignments, control structures and so on, i.e., programming-in-the-small, and object-oriented programming concepts like components, classes, objects, responsibilities and messages, i.e., programming-in-the-medium. This is an important feature of our method, that can be adopted as a holistic approach to building a complex coding curriculum.
- In addition to this methodology, we present a set of tools. The basic building blocks for our toolset are represented by animation actions and animation programs. The metaphors are translated into a library of animation actions over an animation scene. The scene contains the main actors of the animation – like the mechanical arm or the calculator, and their possible interaction. Each animation action animates one basic operation carried out by the processor or virtual machine during the execution of a piece of code, as a set of visible actions of the actors on the scene. In this way, each execution of a target program becomes an animation program on the scene.
- We develop an engine for animation-program execution. This allows to construct and run animation programs offline, or in detached mode. In this mode, the animation program is not directly attached to the target-program source code, and therefore it does not depend directly on the target programming language. This has the advantage of leaving instructors with ample flexibility in choosing programming languages, possibly allowing them to mix and compare programs in different languages.
- In addition to offline mode, we also develop a complex development environment to run animations in online or attached mode. The development environment supports the full cycle of Java source code development and execution, with the important addition of automatically generating animation actions and animation programs for the target code. To do this, we develop a complex infrastructure that represents one of the main technical contributions of our ACME approach.

The ACME methodology and Diogene-CT tools are the product of a long experience with innovative methods for teaching programming conducted over the last 15 years within introductory programming courses at the University of Basilicata. In the last couple of years, the adoption of animation programs has been successfully tested both the “Procedural Programming” course and in the “Object-Oriented Programming” course. This allowed us to gain precious insights into their strength and limitations, and to refine and significantly improve the method.

2. ACME-Approach

We intend to exploit the advantages of earlier approaches within our proposal, while at the same time removing the limitation connected to the scope of the programming platform [6, 7, 8]. We use the two main intuitions from previous work: (a) immediate visualization of the effect of instruction and (b) program as a way to bring the computer to a visually clear target state. But, we also deeply transform them, by constructing our metaphor based on the actual workings of a programming language rather than some fictional microworld or external physical robot.

We develop a methodology and a toolset capable of providing visual feedback for arbitrary programs written using programming languages of state-of-the-art platforms (primarily Java), and both for introductory procedural-programming, and more complex object-oriented programming.

The methodology is based on two different but interrelated metaphors. The first is centered around the main constructs of procedural programming, like storing value by a variable or evaluating a mathematical expression. The second one is constructed around the principles of object-oriented programming, like constructing objects of exchanging messages for method invocation.

The set of tools developed within the project supports different usage scenarios. In fact, Diogene-CT is both a teaching methodology and a set of executable tools. Teachers may decide to only adopt the teaching methodology (the ACME metaphors) as a basis for their lectures, or also the tools that come with it.

The animator tool may be run both in *offline* and *online* mode. In offline mode the teacher typically develops some code, then picks up a usage scenario for it – e.g., calculating the size of the circle given its radius, with a radius of 3 cm – and uses the Diogene-CT animator to give life to the execution of that particular usage scenario, in an offline fashion for the actual execution of the code. Much more challenging is the online mode, in which the Diogene-CT animator is provided with a piece of source code written in Java and allows the student to run the code and interact with it while animating all events triggered by the code.

These ideas are described in the following subsections.

2.1. The Mechanical Arm Metaphor for Procedural Programming

The aim is to teach the basic concepts of any programming language, like variables, assignments, control structures and so on. Based on this idea, we introduce a metaphor for the execution of code instructions based on the use of mechanical arms. A typical scene generated by the animator for this metaphor is shown in Figure 1.

The main elements (characters of the scene) of the metaphor are as follows: (a) The RAM is represented as a large spreadsheet of cells that can be named and can store values, to represent variables; (b) The ALU is exemplified as a calculator capable of performing the computations required in numerical and Boolean algebra; (c) The standard input and standard output are represented by pipes connected to the keyboard and console screen. Animations show streams of characters flowing through these pipes when they are activated; (d) The actions performed by the processor are depicted as movements of a group of mechanical arms. The mechanical arm represents an execution context, typically a function, and is responsible for animating

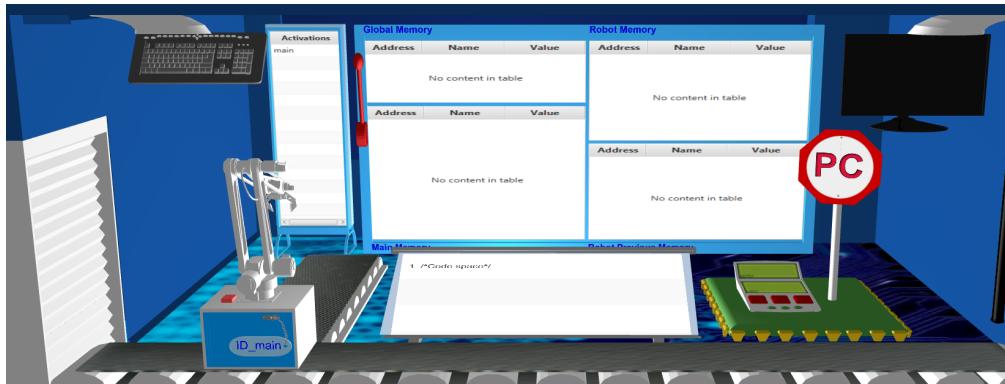


Figure 1: Procedural Programming staging area

operations carried out by the processor or virtual machine, like changing the contents of the memory spreadsheet; operating the calculator to perform computations; activating the pipes to the console or from the keyboard; (e) to highlight the importance of the control flow and the role of control structures, a lollipop sign indicates at any moment of the animation the number of the next instruction that the mechanical arm is supposed to execute. Each instruction causes a set of actions by the mechanical arm on the scenes, guided by the lollipop. Students can see all of the intermediate states of the program animation, and this removes the “black box” effect associated with the traditional execution of code by the machine, and makes it more transparent to students, adding the immediate feedback effect that is typical of robots and microworlds. At the same time, it is highly flexible and can scale from simple, toy examples to complex computations.

The execution of a program consists of a sequence of actions carried out by the machine. The semantics of the program is expressed through the following commands (typically used by the instructors to generate animations in detached mode): show line number and show code line, cell value writer, passing arguments, use calculator, throw an exception, fetch from stdin and push to stdout. To each command correspond an animation of the scene that involves one or more characters.

2.2. The Robot-Community Metaphor for Object-Oriented Programming

The main goal of the project is to bring students to the level of mastering advanced technologies, and primarily object-oriented programming. Experience tells that learning object-oriented method is challenging, even for students that have already acquired basic programming skills. Basic programming – or programming-in-the-small – deals with learning the main instructions offered by the programming language, the techniques to represent data using the program variables, and the process of code development and execution. This is what our mechanical-arm metaphor is focused on. Moreover, object-oriented programming requires that students acquire proper knowledge of programming-in-the-medium techniques, that is, how to organize the code of an application as a set of components – classes and objects, how to assign responsibilities to components, and how to exchange messages among them. In the end, each component will be

described in terms of basic instructions, but the choice and design of components is a crucial design step per se.

We develop a second robot-like metaphor to handle this transition. A typical scene of this second kind is reported in Figure 2.

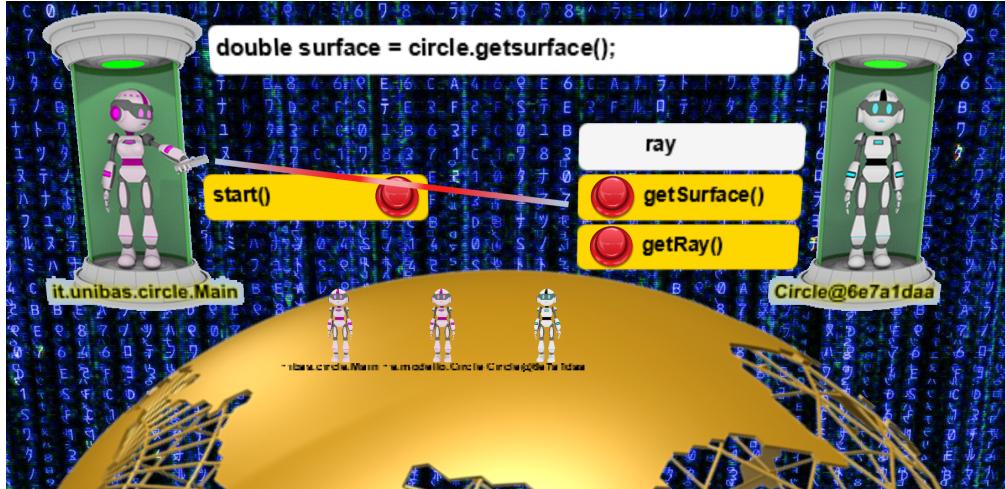


Figure 2: Object Oriented Programming staging area

More specifically, we design a metaphor of object-oriented applications as communities of cooperating robot, each robot being a component (class of objects); the typical concepts of object-oriented programming (reference, message, method, package, binding etc.) will be explained in terms of this metaphor. Every time a class constructor or a static method is executed the robot girl that symbolizes the class is materialized into the holographic tube. It semantically represents a generic OOP class and its animations tend to illustrate the class behaviors: static or objects method invocations, properties usage, and constructors' execution. Thus, the robot boy is generated by the robot girl. Every time a class instance is instantiated or an object reference is used, the robot boy representing the object is materialized into the holographic tube. It semantically represents a generic OOP object and its animations tend to illustrate the object behaviors (i.e.: static or object method invocations and properties usage). The use of the remote control semantically represents the object reference and the exchange of messages. The laser light reaches the opposite part of the scene lighting up methods or properties of the referenced element. The remote control helps learners understand the basic elements of object-oriented programming. The fundamental mechanism of the program execution is the exchange of messages among components. The heap that is represented as a surface on a globe is the element of the metaphor that represents the town where robots live and interact. When a class is first used or every time a new object is instantiated, their little avatar is positioned on the globe. The existence of the little elements allows learners to explore the state of the components and to keep track of the changes. The use of the broom semantically illustrates the garbage collector execution.

The main stage where the metaphor actors play together is composed of background elements that represent the virtual world of robots. The communities of cooperating robots are the places

where classes and objects are picked up and materialized in the holographic tubes. The white rectangle on top is used to show the single running source code line.

As for the robot arm metaphor, there is a list of commands to animate all the Robot-Community metaphor aspects. The semantics of the program is expressed through the following commands: create a new object, zoom to the component, component message, garbage collector and the end component execution.

The Diogene-CT animator is designed to tightly integrate the two metaphors, in such a way that students may animate a piece of object-oriented code to have a high-level view of the cooperative behavior or objects, and at any moment zoom into the behavior of every single instruction by switching to the procedural, mechanical-arm animation to have a low-level view of the execution of the actual instructions.

2.3. Offline - Online Mode

At the core of the project stands the Diogene-CT animator. This is the module responsible for generating the actual visual animation of the code. The animator relies on a library of actions: (a) The action library for procedural programming includes all primitives needed to animate the moves of the mechanical arm, e.g., to change the status of the memory, to use the calculator, to access the keyboard and so on. (b) The action library for object-oriented programming includes the primitives to create new objects, send a message to a class, send a message to an object and so on.

Actions can be composed to form animation programs. The main intuition behind the use of Diogene-CT is to generate an animation program for each execution of the target program. This can be done in two different modes, depending on whether the animator has access to the source code or not.

The simplest usage of the Diogene-CT animator is in offline mode. In this mode, the animator runs separately from the target program, i.e., it does not require to have access to the actual source code. This animator program needs to be built by hand, stored in a file with the appropriate format, and fed to the animator. The animation program is independent from the programming language.

As an alternative, Diogene-CT provides the online or attached animation mode. In this mode, the animator works as a fully-fledged application-development environment, in the sense that it has access to the actual source code. The code can be edited within the environment and run. During execution, the animator captures all events triggered by the target program and animates them on the fly, i.e., it dynamically builds the corresponding animation program. Actions and animations are automatically generated based on the interactive execution of the target program.

3. Conclusions: Lessons Learned

Our experiences with the ACME approach in introductory coding courses, both for procedural and object-oriented programming, have shown that the ACME metaphors and Diogene-CT approach can be highly effective. At the same time, they have provided precious insights about possible developments of the tool.

We typically use quite heavily the tool and the metaphor in the first part of the Procedural-Programming course. This part usually covers approximately 40 hours of teaching, with 2:1 lessons to lab ratio, and goes from introductory concepts, like variables and data types, to functions, modular programming, arrays and lists. In the course of these initial lessons, the animations give students a very concrete perspective on the actual operational semantics of instructions, and allow instructors to discuss in great detail both simple concepts – like variables or parameter passing – and more complex ones, like recursion or pointers. The tool, when used in attached mode, also greatly helps student in debugging their first attempts at solving simple exercises, by providing a clear and graphical depiction of their logical errors.

However, we have noticed that, as soon as students become more experienced with coding and start dealing with medium-complexity tasks – like developing menu-based user interfaces, or writing algorithms on collections – they tend to rely less on the tool. In fact, the robot-arm metaphor may become boring when applied to very large codebases, since its perspective is very fine-grained, and therefore medium and upper-level students may get the impression that it slows them.

To solve this problem, in the second half of the course we tend to use the tool essentially in detached mode. We record videos of long animations of medium-complexity code, and then cut the full videos to extract the most significant parts that can be useful to students. In this respect, we believe that it would be very useful to introduce breakpoints, in the style of debuggers, to enable users to fast-track to specific points within the code, and a trace-like function to start and stop the generation of animations at these points.

The tool is perceived as a very useful addition again as soon as students are exposed to object-oriented programming. The introduction to objects and classes usually covers approximately 20 hours of our Object Oriented programming course, also in this case with a 2:1 lessons to lab ratio. Throughout these lessons, the possibility of exploring code executions by looking at the sequence of messages and the heap state is typically appreciated by students. Similarly, the tool be very useful when introducing concepts like object hierarchies and binding. In these cases, Diogene-CT can be used to emphasize the implicit association between object in a hierarchy through the “super” property.

Again, when projects become more complex and the heap increases in size the tool tends to become somehow less effective. Also in this case, debugger-like functionalities would very useful in order to restrict the collection of objects to focus animations on.

It is important to emphasize that not all important concepts can be properly addressed using Diogene-CT alone. For example, methodological guidelines like information hiding, the notion of interface vs implementation can only indirectly be emphasized during animations. Similarly, the comparison of strictly algorithmic aspects, like, for example, hashing vs sorting, is not among the main goals of Diogene-CT.

In this respect, we believe that Diogene-CT can be effectively used in conjunction with other tools, like, for example, algorithm visualization tools or UML-based code-visualization tools to complement and reinforce their abstractions.

In conclusion, ACME methodology and the associated tool Diogene-CT can support teaching and learning computer-programming skills in school curricula of the secondary grade and higher. Diogene-CT represents the first of a new breed of approaches, that we called code-animation environments.

We also conducted some experiments and users study [5] at the University of Basilicata. Based on that experience we obtained largely positive feedback, thus confirming the effectiveness of the approach.

References

- [1] S. Marginson, R. Tytler, B. Freeman, K. Roberts, Stem: country comparisons: international comparisons of science, technology, engineering and mathematics (stem) education. final report. (2013).
- [2] G. Siekmann, P. Korbel, Defining stem skills: review and synthesis of the literature, in: NCVER, Commonwealth of Australia, 2016.
- [3] H. Watt, Session d: Promoting girls' and boys' engagement and participation in senior secondary stem fields and occupational aspirations (2016).
- [4] J. M. Wing, Computational thinking, Communications of the ACM 49 (2006) 33–35.
- [5] G. Mecca, D. Santoro, N. Sileno, E. Veltri, Diogene-ct: tools and methodologies for teaching and learning coding, International Journal of Educational Technology in Higher Education 18 (2021) 12. URL: <https://doi.org/10.1186/s41239-021-00246-1>. doi:10.1186/s41239-021-00246-1.
- [6] M. U. Bers, L. Flannery, E. R. Kazakoff, A. Sullivan, Computational thinking and tinkering: Exploration of an early childhood robotics curriculum, Computers & Education 72 (2014) 145–157.
- [7] B. Harvey, Computer Science Logo Style: Symbolic Computing, volume 1, MIT press, 1997.
- [8] H. Tsalapatas, O. Heidmann, R. Alimisi, E. N. Houstis, Game-based programming towards developing algorithmic thinking skills in primary education, 2012.