The Journal of Systems & Software 195 (2023) 111516

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

EUDability: A new construct at the intersection of End-User Development and Computational Thinking☆

Barbara Rita Barricelli [a], Daniela Fogli [a,*], Angela Locoro [b]

[a] University of Brescia, Department of Information Engineering, Via Branze 38, Brescia, 25123, Italy
[b] University of Insubria, Department of Theoretical and Applied Sciences, Via O. Rossi 9, Varese, 21100, Italy

ARTICLE INFO

Article history:
Received 1 June 2022
Received in revised form 2 September 2022
Accepted 19 September 2022
Available online 27 September 2022

Keywords:
End-User Development
EUDability
Computational Thinking
Digital transformation
Life-long learning
Sustainable workplace

ABSTRACT

The sustainable and digital future of work may imply a dramatic equilibrium change between social factors and technological ones. We argue that providing suitable tools to support End-User Development (EUD) in the workplace could represent a way to cope with such future changes. The contributions of this paper include the analysis and characterization of the most used EUD techniques and their crossover with a new conveyed model of Computational Thinking. The synthesis between these aspects is made explicit in the construct of EUDability, which is designed to capture the quality dimensions of EUD systems suitable to work scenarios where better roles and better tools for individuals may be shaped. EUDability has to do with identifying and assessing the difficulties of EUD techniques on one side and the Computational Thinking skills held by individuals on the other side.

1. Introduction

To facilitate the construction of a truly sustainable future, we need to train the citizens of the future to ensure that they possess the necessary skills to interact with a world that will become more and more digital. To this aim, extensive literature in education promotes starting coding as early as the K-12 curriculum (Barr and Stephenson, 2011a; Fletcher and Lu, 2009; Grover and Pea, 2013). However, this change will not only affect the younger generation but also may find unprepared those who are already working (the age group 35–60) and who often do not possess the adequate kind of literacy (DiSessa, 2018) to cope with the changes that are expected.

Several studies state that, since now and in the next decade, there will be a high job turnover, which implies that many workers will be replaced by machines and other ones will have to cope with them in an ever-stricter sense than before; moreover, there will be a high loss of jobs in certain areas, which will be replaced by other jobs with higher added value (PwC, 2018; World Economic Forum, 2020). In particular, workers will be demanded to do fewer manual tasks in favor of problem-solving activities that require more information and knowledge management (Kaasinen et al., 2019; Merisotis, 2020), and the ability to develop, extend and customize IT systems (Costabile et al., 2008; Manca et al., 2021; Meister and Brown, 2020). Thus, work complexity will increase for all workers who do not have any of those computational background and skills that are deemed necessary to deal with the current and the forthcoming complexity of work environments (Kusmin et al., 2018).

Changing mindset and practices in the workplace in this direction requires developing tools within workers' reach, which should not expect advanced Computational Thinking skills but eventually gradually favor their acquisition. We believe that the End-User Development (EUD) approach (Lieberman et al., 2006; Paternò and Wulf, 2017), and the techniques that have been proposed and refined in this field of research in recent years (Barricelli et al., 2019), may fit as the right support and aid for achieving what work transition has in store for us. EUD aims to empower end users to create, adapt, and extend digital artifacts at a level of complexity that is adequate to their background, skills, and preferences (Lieberman et al., 2006). Therefore, it not only includes features for system modifiability but also encompasses domain-specific environments for the development of new digital artifacts. Hence, EUD can be considered a tool for supporting the social and labor sustainability of the ecological, energetic, and demographic transition.

EUD is one of the fewer fields devoted to understanding who is on the other side of the technology before designing it, i.e., humans. Nevertheless, an issue that is often overlooked is the dual aspect of EUD, that is, the task/concept/tool difficulty on one side and the individual's ability on the other side. Such dual aspect

☆ Editor: Xiao Liu.

* Corresponding author.
E-mail addresses: barbara.barricelli@unibs.it (B.R. Barricelli), daniela.fogli@unibs.it (D. Fogli), angela.locoro@uninsubria.it (A. Locoro).

https://doi.org/10.1016/j.jss.2022.111516
0164-1212/

can be addressed through a method that pertains to either the users' ability to pose and solve problems or the difficulty of the techniques to help develop a solution and the effectiveness of the tools to enable reaching a preferable state from a problematic one.

The construct usually adopted to assess the suitability of EUD solutions for people's comfort is *usability*. Although it is a quite standardized construct, still it is deemed very general-purpose. It is not explicitly dual in nature and tends to capture only limited aspects of the motivations behind EUD design: that of providing the right intersection between the difficulty of using a tool for EUD purposes and the ability of an individual to use it.

Therefore, the research question we would like to investigate in this work is: *How can tool difficulty and user's ability be reconciled in the design and evaluation of EUD solutions?*

The socio-technical perspective adopted in the meta-design approach to EUD partially responds to the question. Meta-design has been deeply discussed in Fischer and Giaccardi (2006) and then revisited in Fischer et al. (2017); it is conceived as a socio-technical framework supporting the creation of open systems that can evolve at use time in the hands of end users. Literature on meta-design does not include operative dimensions for development and evaluation of such open systems, but proposes a concept at a higher level of abstraction that helps embrace several situations of user participation (Barricelli et al., 2016). We argue that the duality between the difficulty of using a EUD tool and the worker's ability to deal with it may be accounted for through an explicit characterization, that is, a new construct able to catch and consider both aspects together in a more operative and practical manner with respect to meta-design.

To answer the above research question and make more concrete and operative the concepts behind EUD, this paper proposes a model, based on the literature, for characterizing Computational Thinking (CT) and the related skills, which helps frame the dual, human-sided counterpart of the technology used in EUD scenarios. This model will help to match EUD techniques mostly practiced in work scenarios with the ability of users to exploit them. These work scenarios currently do not require or require limited use of personal computers. In so doing, we shed light on the level of CT skills they may request and help predict the potential and suitability of EUD techniques in a given context within a given target of workers. The paper then proposes a definition of EUDability: a novel construct encompassing several dimensions that, together with CT skills, account for the correct use and deployment of EUD tools in the workplace, as well as for their assessment.

To this aim, we first describe EUD and the results so far reached in this research domain, we discuss the work scenarios where EUD can be applied, and we classify the most used EUD techniques in an original way (Section 2). We then propose a process model for CT based on the definition of CT skills to be mapped with EUD techniques (Section 3). In Section 4, we identify six EUD techniques and describe their crossover with the CT model by figuring out specific application scenarios. In Section 5, we define EUDability and discuss its relevance for the evolution of EUD and its potential in shaping a sustainable future of skilled workers. Section 6 summarizes the contributions of the paper, the limitations of the research carried out so far, and the next steps for future work.

2. EUD techniques emerging from the interplay of paradigms and interaction styles

2.1. Research background

Since 2003, when the European Network of Excellence on End-User Development (EUD-Net) was established, EUD evolved thanks to the work carried out by researchers all over the world. The first definition of EUD was published in 2006 in a EUD-Net curated book (Lieberman et al., 2006): "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact". This definition binds the concept of EUD to the creation of software artifacts alone, while today, thanks to the spread of Internet of Things (IoT), the potential for the application of EUD methods and tools expands much further. Hence, a new definition has been proposed in Barricelli et al. (2019), which describes EUD as "the set of methods, techniques, tools, and socio-technical environments that allow end users to act as professionals in those ICT-related domains in which they are not professionals, by creating, modifying, extending and testing digital artifacts without requiring knowledge in traditional software engineering techniques". What the two definitions have in common, and characterizes the End-User Development discipline, is the interest in supporting the evolution of the end users from passive consumers to active producers of software artifacts (Fischer, 2002).

EUD has been and still is applied to various application domains — e.g., healthcare, business and data management, web applications and mashups, smart objects and smart environments, games and entertainment, education and teaching. When the context of application is an organization, end users are peculiar subjects in that they are experts, not in computer science or IT in general, but in the domain in which they work; this makes them fully aware of the needs, open problems and challenges relevant to their work environment and their profession. The application of EUD in the workplace has been studied since the very early days of EUD and even before the discipline had a proper name: in Mørch et al. (2006) the authors describe regular users as those workers who are not developers and not interested in tailoring but want to use computers to perform they daily work; Nardi and Miller (1990) define the non-programmer users as workers who could also have programming skills. One of the most important aspect of designing and developing EUD tools is to allow end users to create programs following their reasoning habits, and not the computing habits of software developers embedded in traditional programming languages — e.g., Repenning and Ioannidou (2006) and Whiley and Blackwell (2001). This means building interactive systems that behave in ways that are familiar to end users and help develop their reasoning and achieve their goals.

Several survey papers analyze the scientific literature on EUD. In Paternò (2013), the author identifies a set of key concepts that characterize the research in the EUD field, such as the need to design EUD tools to balance application complexity and learning effort required and the importance of providing users with environments that allow them to manipulate content more than forcing them to use scripting languages (e.g., macros, formulas). He also reflects on three main dimensions that can be used to compare EUD applications: the generality of the approach (if it can be applied to various domains), the coverage of main interactive aspects (if both interactive and functional parts are addressed), and the presence of abstractions for hiding implementation details. The findings of the review published in Tetteroo and Markopoulos (2015) suggest that most of the analyzed literature is focused on the engineering of systems and laboratory evaluations, while action and basic research material are lacking. The authors specifically underline the potential of action research for the evaluation of EUD systems in natural environments and that its absence can be linked to a lack of understanding of what causes the engagement of users in EUD activities. The work of Maceli (2017) extends the previous work and identifies 13 categories of EUD technology tools. Among these, the most prevalent are "programming environments and frameworks", "web and information authoring tools", "mashup tools",

and "spreadsheets". An important finding of Maceli's survey is that "little work addressed newer user interface paradigms such as tangible or voice interfaces". Hang and Zhao (2015) present a systematic review focused only on end-user service composition. Specifically, the authors classify their results according to five main activities: service composition, service design, service reuse, service testing, and debugging. Among these activities, the most supported one is composition, followed by reuse and design, while just a few works address testing and debugging. The systematic mapping study (Barricelli et al., 2019) analyzes papers about EUD, End-User Programming (EUP), and End-User Software Engineering (EUSE) and presents the results of the literature analysis according to seven dimensions: type of approach, type of technique, phase in which the shaping activities take place, application domain, target use, classes of end users, and empirical validation. The classification based on the type of technique does not make a distinction between interaction styles and EUD techniques, mixing them together. The present paper refines this kind of classification. Finally, Ponce and Abdulrazak (2022) focus their literature review on EUD, EUP, and EUSE techniques to develop context-aware applications. Metaphors and interaction styles of such techniques, as well as their implementation approaches, are analyzed. What emerges from this study is that the most common implementation approach is rule-based programming.

2.2. EUD in the workplace

The empowerment of end users, specifically of domain experts in their work environment, is something that has been studied for a long time in the fields of Computer Supported Collaborative Work (CSCW) (Grudin, 1991) and Human Work Interaction Design (HWID) (Abdelnour-Nocera et al., 2015). The latter, in particular, focuses on the integration of work analysis and interaction design methods. In conjunction with EUD, HWID helps support the introduction of new technologies for establishing pervasive and smart workplaces (Valtolina et al., 2017; Barricelli et al., 2015).

In this light, we focused our attention on scenarios where workers are called on to interact with IT systems, including hardware and software components, such as robots, Internet of Things (IoT), and healthcare devices. These systems are usually complex and safety–critical; therefore, they are designed and developed by engineers and programmers according to the requirement analysis carried out with customers and users. Modifications and extensions of these systems usually require the intervention of initial designers and experts, and, also for this reason, they are not so frequent. Nowadays, the situation is changing, and scientific methods are proposed to empower workers and let them tailor those systems for adaptation to the progressively changing requirements.

Here, we present three work scenarios where workers do not usually perform EUD activities, and we propose for each of them possible EUD approaches that might be suitable. This selection intentionally excludes all scenarios where workers are already called on to carry out some form of EUD, e.g., by manipulating spreadsheets or using authoring tools, as in business data management, game development, education, interaction design, and communication management. Indeed, workers in these domains are dealing with digital tools and are naturally more skilled in data manipulation, creating macro-like solutions, or composition of graphic user interfaces (Barricelli et al., 2019).

The first work scenario involves the interaction with collaborative robots. Collaborative robots are becoming more and more affordable for small and medium enterprises to address the current requests for mass customization and production flexibility. These robots are a new kind of industrial robots that can share the workspace with human workers and collaborate with them by performing repetitive, unsafe, and precise tasks, leaving to the workers' activities that require problem-solving, decision making, and uncertainty management (Schou et al., 2018; Parsa and Saadat, 2021). An important mid to long-term expectation from collaborative robots is that they could be instructed directly by workers to perform the requested tasks without the need to resort to professional programmers (Hamabe et al., 2015; Huang and Cakmak, 2017; Fogli et al., 2022). Some robot producers have started to provide operators with walk-through programming features that support easy programming of tasks by directly moving the robot's end-effector in the desired positions and recording the trajectory points and the joint coordinates. An extension of these features was proposed in the literature, where the robot is able not only to repeat but also to learn different movements under different conditions and then generalize them. This technique is programming-by-demonstration (Alexandrova et al., 2014; Hamabe et al., 2015), and relies on the same idea underlying the seminal work of Dey et al. (2004) for EUP in context-aware applications managing meetings and medicine taking. A second proposed technique for robot programming by end users is the component-based technique. The worker interacts with a visual programming environment in which graphic blocks (usually puzzle-like) are combined to create the desired program. Code3 (Huang and Cakmak, 2017) and CoBlox (Weintrop et al., 2017) are examples of EUD environments exploiting this technique.

The second work scenario where workers will be called on to shape and personalize the behavior of a complex IT system is that of smart environments and IoT ecosystems. Industrial IoT applications, ambient-assisted living systems, and smart heritage sites and museums are examples of contexts where IoT technology can be employed and customized by domain experts (e.g., manufacturing workers, caregivers, and cultural heritage professionals, respectively). Manca et al. (2021) outlined that the behavior of this kind of systems cannot be completely coded at design time by professional developers since they often lack the domain knowledge necessary to address all possible situations that may occur at use time. Also, in these cases, different EUD environments have been proposed. Most of them exploit a rule-based technique that allows end users to perform Trigger-Action Programming (TAP), that is, to define new behaviors of an IoT ecosystem in the form of IF-THEN rules. This usually means selecting a possible event and/or condition from those available in a rule editor, to define the IF part of the rule (trigger), and then selecting the action(s) that must be executed when the event occurs or the condition is satisfied. For instance, Manca et al. (2021) applied a trigger-action approach to a paper factory; to this aim, the authors extended an existing EUD platform to provide triggers and actions relevant in that specific industrial context, such as triggers/actions related to different worker roles, key environments of the factory, and sensors and actuators available in the production lines. The TAP approach has also been adopted for the creation of smart visiting experiences in cultural heritage contexts (e.g., Ardito et al., 2018; Fanni et al., 2019). Ardito et al. (2018) proposed using a rule editor to enable cultural heritage experts to define the behavior of IoT devices available in the cultural heritage site. Fanni et al. (2019) presented a graphical authoring tool based on TAP that supports domain experts in developing point-and-click video-games for promoting cultural and environmental heritage.

In the third scenario, healthcare professionals represent a further category of workers who may be called on in the future to create and customize digital applications and smart objects for their patients. Tetteroo (2017) designed and evaluated a platform for physical rehabilitation after strokes, multiple sclerosis, and spinal-cord injuries. The platform includes a EUD visual programming environment adopting a component-based approach, which

supports physiotherapists in creating and modifying exercises exploiting an interactive board and physical objects endowed with RFID tags. A component-based technique is also adopted in Barakova et al. (2013), which proposes a EUD environment that helps define behaviors of social robots to be used in therapies with autistic children. Barricelli and Valtolina (2017) presented a rule editor to support trainers of non-professional teams of athletes monitoring and analyzing health data streams coming from athletes' wearable devices.

2.3. Characterization of EUD techniques

According to the systematic mapping study of Barricelli et al. (2019), the most adopted EUD techniques are: *component-based*, *rule-based*, and *programming-by-demonstration*. As shown in the previous sub-section, these techniques often match users' needs in the work scenarios considered in this paper.

Going more deeply into their characterization, we can conceive them as different EUD paradigms, similarly to the classification of programming languages in the related programming paradigms (e.g., procedural, functional, declarative, and object-oriented).

Such EUD paradigms can be applied individually or in combination, and are implemented in EUD environments by offering a specific interaction style. Traditionally, the HCI discipline confronts direct manipulation with the command-based interaction style. Direct manipulation in graphical user interfaces (GUIs) is derived from the technological evolution provided by the bitmap display and the mouse device. Nowadays, direct manipulation can also be physically performed on different objects, such as robots and the variety of smart devices available on the market. The command-based style has recently evolved into the conversation-based style adopted in chatbots and smart speakers. Therefore, it is not rare that the considered EUD paradigms are implemented by offering different interaction styles, which can, in turn, influence the intuitiveness and learnability of the EUD technique.

The component-based paradigm foresees the composition of digital entities that may represent programming statements or, more simply, domain-specific concepts. The composition activity is usually performed in a GUI through a direct manipulation interaction style according to a metaphor that yields the creation of jigsaw puzzles, box-and-wire networks, flowcharts, and so on. In some cases, the building blocks of a program are created and composed using a different interaction style; for instance, in Fogli et al. (2022), a conversational interface is provided to create a robot program by composing entities that represent robot actions, objects to be manipulated and locations where the objects can be placed.

The rule-based paradigm is adopted in all of those situations where users would like to define new behaviors of a system. For instance, a user could create IF-THEN rules representing the behaviors of a smart home (e.g., IF it is Sunday and 9.30 a.m. THEN raise the blinds; IF the garage sensor detects a movement THEN send me a message; etc.). Starting from the well-known IFTTT (*If This Than That*) application (Ur et al., 2014), most of the EUD environments supporting the rule-based paradigm provide a GUI interface with direct manipulation (Ardito et al., 2018; Barricelli and Valtolina, 2017; Ghiani et al., 2017): usually, the user defines the trigger by selecting events and conditions from an available list, and then selects from another list those actions that must be executed when the trigger fires. Recently, conversational interfaces like chatbots (Asunis et al., 2021; Stefanidi et al., 2019), interfaces for message exchange (Huang et al., 2019), and speech-based interfaces (Chkroun and Azaria, 2019) have been developed and experimented to support rule creation, thus introducing an interaction style for the rule-based paradigm alternative to direct manipulation.

The programming-by-demonstration paradigm is based on the observation of the user's actions: the user demonstrates the execution of a task by performing actions on example data; the system records these actions and then infers a generalized program that may work on new data. Task execution can be performed on a GUI through direct manipulation (Dey et al., 2004) or on a physical device like a collaborative robot (Alexandrova et al., 2014); alternatively, the user could describe the task by voice or text in a conversational interface, and the system behaves like a learner that listens to a teacher's lesson: for example, Li et al. (2017) propose a mobile application that combines natural language understanding and programming-by-demonstration to support end users in the creation of smartphone automation.

Given these considerations, our analysis of EUD techniques covered two dimensions: the paradigm and the interaction style. This analysis led to the identification of six techniques, schematized in Fig. 1, which will be analyzed in Section 4 and crossed over with the Computational Thinking model discussed in Section 3.

3. Computational thinking for EUD: a comprehensive model

3.1. Research background

Computational Thinking (CT) key aspects, and the related skills applicable to the domain of EUD, emerge from many literature surveys on the topic.

One of the first, overall, and largely adopted definitions of Computational Thinking (CT) dates back to Wing (2006) and states that CT "involves solving problems, designing systems, and understanding human behaviors, by drawing on the concepts fundamental to computer science". Digging into the details, elements, steps, and processes involved in CT was the main aim of the several studies characterizing CT that, since 2006, were published to influence the direction and pathways taken by educational and technological fields in exploiting the CT concept. Wing soon refined its definition to be more focused on the CT primary processes, those of "formulating problems and their solutions, so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing, 2011). This refinement also moved from the many other studies that tried to define, eviscerate and operationalize the necessary steps towards the characterization of CT. From 2011 (Barr and Stephenson, 2011b) up to 2019 (Pollak and Ebner, 2019), recurrent aspects of the CT problem-solving cognitive process were identified (Brennan and Resnick, 2012; Selby and Woollard, 2013; Moreno-León et al., 2015; Turchi et al., 2019). They can be summarized as follows:

- understanding of what parts of a problem may be subject to a computational process;
- identification of the right tools for all the computational parts of the problem;
- automation of the process to treat those parts with the identified tools;
- checking that everything is correct;
- learning to reuse the same solution for other or unseen problems.

In 2021, a new definition of CT by Wang et al. (2021) was introduced in STEM (Science, Technology, Engineering, and Mathematics) education. CT was then seen as a manifold process made of "extracting key information from the concrete details of a problem (abstraction), reformulating a larger problem into a set of smaller ones (decomposition), detecting the patterns embedded in data, developing and applying algorithms, and so forth". The many activities and abilities mentioned in this new

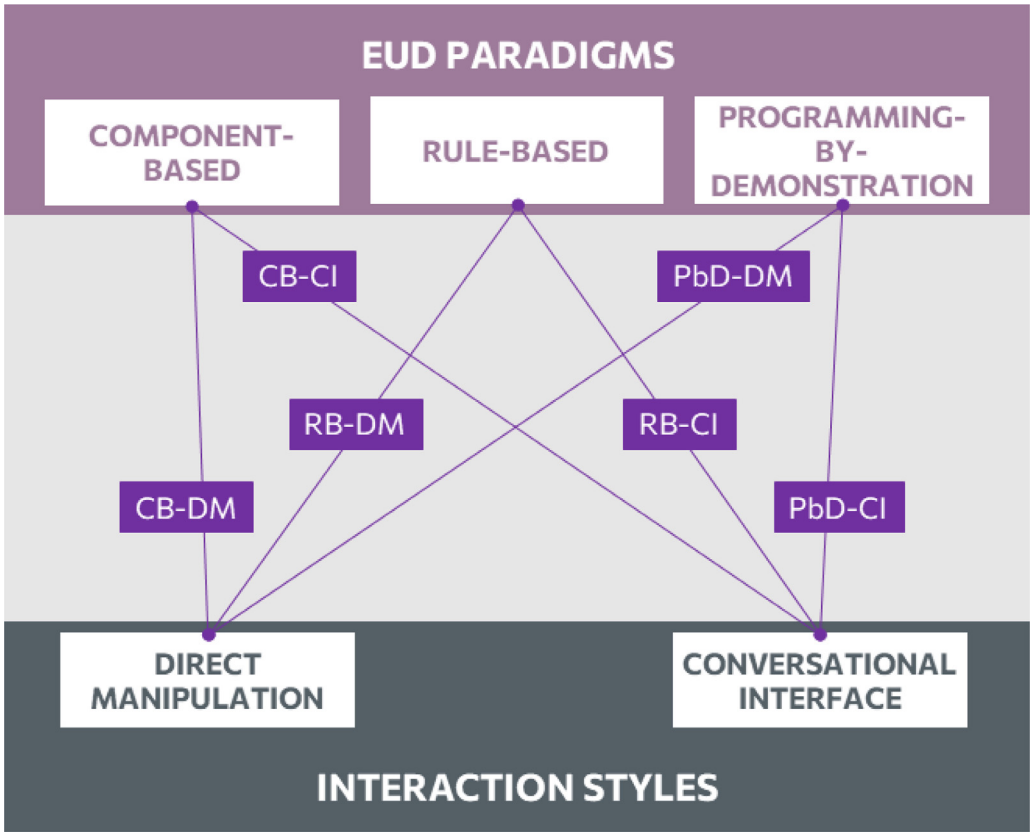

Fig. 1. The six EUD techniques. The purple boxes contain acronyms of the paradigms and interaction styles whose extended terms are reported in the white boxes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

definition were stemmed and merged from both the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) for K–12 education, who summarized CT primary tasks as follows: "CT is a problem-solving process that includes formulating problems, using a computer or other tools, logically organizing, analyzing, and representing data, automating solutions through algorithmic thinking, achieving efficient and effective solutions, and generalizing and transferring to other problems".

Distilling the CT skills associated with the above-defined problem-solving process brought heterogeneous outcomes. One of the most recent and widest studies on the matter, Hsu et al. (2018), included all of the CT skills mentioned in the previous studies and definitions.

A noticeable advancement in the operationalization of the CT concept was that of characterizing CT skills with the aim to assess them. The first validated attempt in this direction was that of Román-González et al. (2017), relying on the very first definitions and lists of CT processing and mental stages. The Computational Thinking test (CTt) is the outcome of this process. One of the most recent and comprehensive works in this respect is that of Tsai et al. (2021). The authors devised a Computational Literacy Scale (CLS) of nineteen items, which are classified according to five subscales: Abstraction, Decomposition, Algorithmic Thinking, Evaluation, and Generalization. The abstraction subscale aims to assess the skill of selecting the main information to solve a problem. The decomposition subscale aims to assess the capacity to break problems into smaller ones. The algorithmic thinking subscale aims to examine how the individual can organize a step-by-step solution to the problem. The evaluation subscale aims to assess the level of optimization of the solution. The generalization subscale aims to identify the ability of the individual to recognize similar patterns of the present solution to be applicable in similar future problems.

Surprisingly, only one framework was developed to encompass both CT mental stages and the related dimensions (Palts and Pedaste, 2020), though such dimensions did not converge to skills. According to this framework, the mental stages of the CT processes are: (i) defining the problem, (ii) solving the problem, and (iii) analyzing the solution. The first stage includes all CT steps needed before starting to solve the problem. Abstraction is deemed essential to identify and extract main ideas, which is related to modeling. At this stage, also reformulating the problem may be useful. The second stage implies the decomposition of the problem into smaller pieces. The next steps are related to the design of the solution and include algorithm design (including, in its turn, data collection, automation, simulation, ordering, and the like). An important last stage is the solution analysis, which includes the dimensions of generalization and evaluation.

3.2. An operational model of computational thinking

The literature analysis highlighted the existence of differences among Computational Thinking aspects. Nonetheless, we were able to discern some common aspects of the Computational Thinking process and relate them to the EUD process: the attitude to solve problems (CT), which comes by articulating activity stages, and by practicing CT skills. As to the activity stages, the three-stage mental process of Palts and Pedaste (2020) appeared to be the most suitable to describe the EUD process, according to our own experience about the design of EUD environments. Then, the five basic CT skills adopted in the CLS of Tsai et al. (2021) emerged as recurrent in literature proposals, as well as fundamental in using the EUD techniques characterized in the

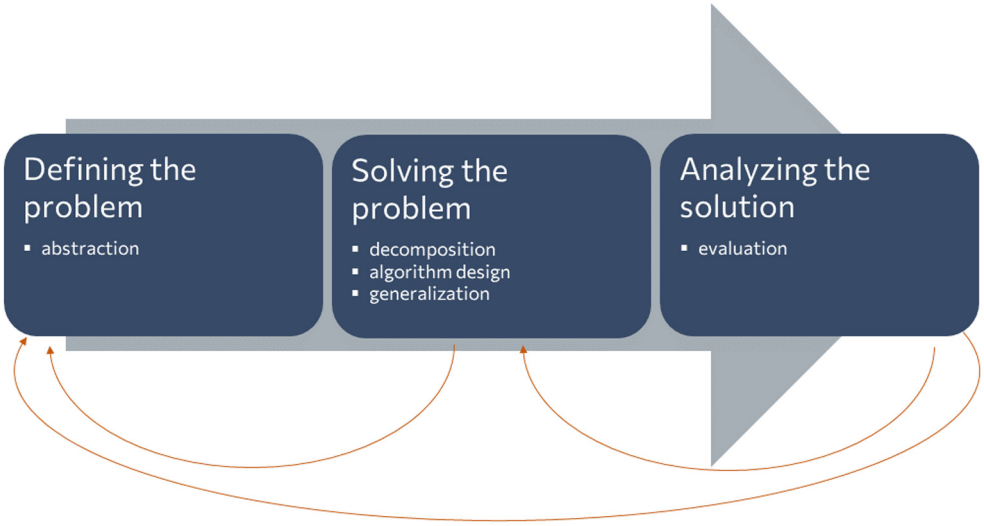

Fig. 2. A Computational Thinking Model for EUD.

present work. In this light, operating with a EUD tool may be affected by the level at which anyone is comfortable with using that tool, which in turn requires CT skills reached through a *degree of individual mastery* that allows anyone to manage such tool comfortably.

From the above analysis and considerations, we derived the following operational definition of Computational Thinking for EUD: *the acquired capability of adopting a three-stage mental process, i.e., defining the problem, solving the problem, analyzing the solution, by knowing how to apply five basic skills, i.e., abstraction, decomposition, algorithm design*[1] *generalization, and evaluation, up to an individual level of mastery*.

The schematization depicted in Fig. 2 shows the three stages of our CT model, how they are related, and how they can be articulated, that is, in which direction each of them may follow or come back to the previous one. Each stage informs the following one, and, in the end, the process may restart. Notwithstanding the main flow of the stages, a back and forth performance may be put in place in real settings. This three-stage model also shows when the Computational Thinking skills are activated and applied.

These skills, borrowed from the theory of the five mental processes of Tsai et al. (2021), can be regarded as *the abilities of individuals to express, activate and apply each mental process necessary to understand, execute and complete each stage of the problem identified. They constitute the degree of ability of individuals.* In particular, they are characterized as follows:

- *Abstraction* refers to the cognitive process of focusing on the key information rather than the details of a problem to be solved;
- *Decomposition* refers to the cognitive process of identifying the main parts into which a problem may become a set of smaller (and more tractable) problems;
- *Algorithm design* refers to the cognitive process of planning the necessary steps for defining the solution through a step-by-step procedure;
- *Generalization* refers to the cognitive process of recognizing problem-solving patterns in other problems and reuse the same (possibly adapted) solution, and apply it to unseen and unsolved future problems;
- *Evaluation* points to the cognitive process of assessing the correctness of the solution and of comparing it with other ones, in order to optimize it, given the resources at one's disposal.

As shown in Fig. 2, the application of CT skills may be repeated one or many times until convergence is reached, and the principal flow may advance. For example: decomposing the problem into sub-problems may bring to a deeper level of detail, which may require further abstraction activities; counterfactual analysis may bring to refine the decomposition, algorithm design and generalization instances; and so forth.

Finally, each stage and the related skills are operationalized in our model as follows:

1. *Defining the problem*, i.e., creating a model through the selection of the main features and their abstraction. Note also that a problem is a class of questions, not a single instance, e.g., like in $n * m$, where abstraction is obtained through mapping concrete elements to formal class symbols. This stage corresponds to the skill of abstraction, i.e., the ability to understand, interpret, and translate the main aspects of the problem (a system of concrete and informal signs) into a formal language (a system of symbols or formal signs);
2. *Solving the problem*, i.e., decomposing the problem into elementary pieces and identifying a step-by-step procedure to solve them partially (and locally) and then globally (all its instances) — this stage corresponds to the skills of decomposition (i.e., breaking the problems into sub-problems), algorithm design (i.e., systematize the passages to manipulate, order and compute the solutions to each sub-problem into regularities, rules, and their composition), and generalization (i.e., recognizing regularities, rules, and patterns and/or creating new ones that are reusable in order to provide a new system of mappings from concrete signs into formal signs);
3. *Analyzing the solution*, i.e., checking whether the solution is sound, correct, complete, and the like; this means executing the procedure and finding whether it reaches the goal effectively and efficiently — this stage foresees the application of the CT skill of evaluation (i.e., providing counterfactuals, simulations, informal or formal testing, deriving heuristics, rules of thumb, and sound judgments and proofs).

[1] We use the term *algorithm design* instead of *algorithmic thinking* to avoid overlapping with the term *computational thinking*.

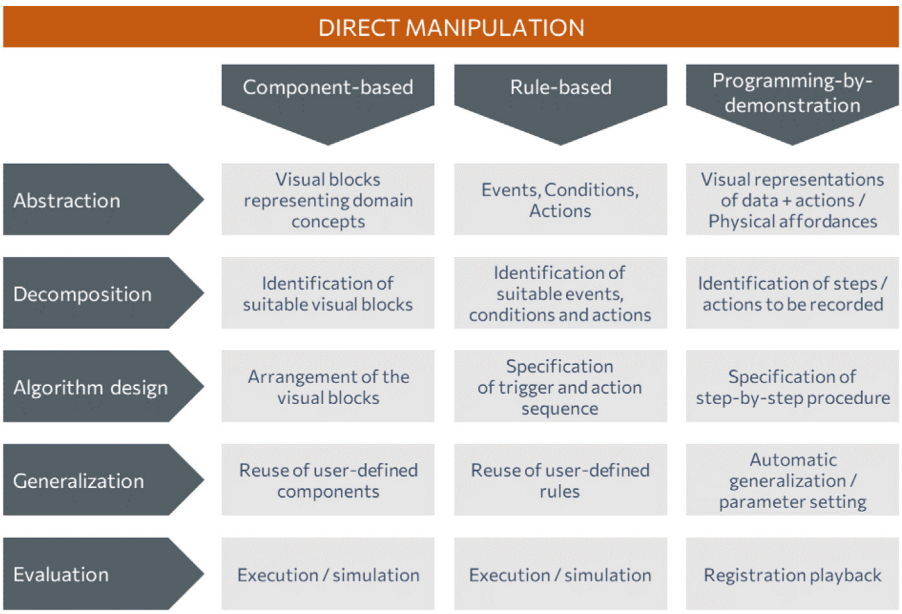

Fig. 3. Considering the direct manipulation style, each box shows a concept that characterizes the intersection of one of the three EUD paradigms considered in this paper and one of the Computational Thinking skills identified in Section 3.2.

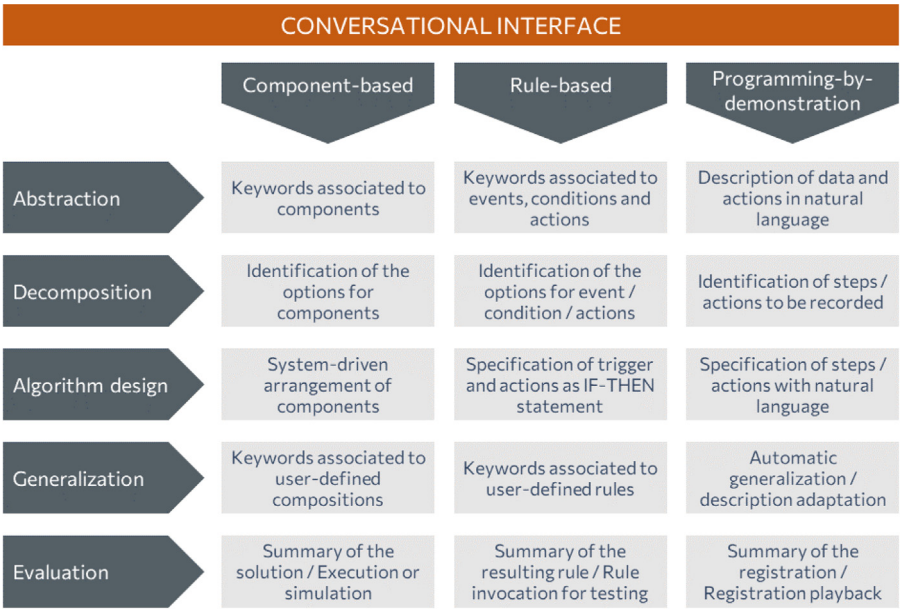

Fig. 4. Considering the conversational interface, each box shows the concepts that characterize the intersection of the three EUD paradigms and the Computational Thinking skills identified in Section 3.2.

4. Crossover of EUD techniques with computational thinking

In this section, we describe each EUD technique in Fig. 1 with respect to Computational Thinking and the related CT skills. We argue that this mapping could bring a double-fold benefit: understanding the level of ability that a worker adopting that technique should possess; and understanding how the proper design, implementation, and continuous use of the technique could take workers' ability into account and possibly increase it. Figs. 3 and 4 schematize the key concepts related to the intersection of the EUD techniques characterized in Section 2.3 with the main interaction styles, i.e., direct manipulation interaction style and conversational interaction style, respectively.

As highlighted in the two figures, our analysis was carried out by giving precedence to the choice of the interaction style, as the designer, first of all, has to decide whether a system with direct manipulation interaction or one with a conversational interface is more suitable to the specific case and its context. The choice is guided by the designer's experience and other factors affecting the work environment (e.g., the possibility for the workers of using their hands to interact with a screen, the noise level of the environment, and the like).

The following sub-sections describe how individuals can activate and apply each CT skill to understand, execute and complete each stage of a EUD problem, which may consist of the adaptation, extension, or creation of a digital artifact.

4.1. Component-based and direct manipulation (CB-DM)

The analysis of the technique that combines the component-based paradigm with direct manipulation, with respect to the CT model, leads to the following considerations:

- *Defining the problem*: components are often linked to visual blocks representing concepts of the application domain and are abstracted away from the technical details of how they work and of how data is passed from one component to another; such components are usually listed in different libraries. Thus, no effort should be required to create the problem model. Difficulties could arise whenever the components are not at the right level of abstraction that the user can comprehend, for example, when components are represented as graphic symbols associated with specific programming structures (loops, conditionals, and so on);
- *Solving the problem*: the next step is that of identifying the components that are suitable to the case in question, including control and iterative components. Choosing the correct components could require some basic knowledge of (or minimal training on) the composition tool. Components usually have inputs and outputs to connect multiple components in sequence, and some components may represent repetitions of actions. Defining proper connections can be facilitated by the form or color of the blocks (e.g., jigsaw puzzle pieces) or by using specific wires. Direct manipulation may support combinations of components by dragging-and-dropping: the approach, therefore, usually allows one to plan a solution with step-by-step procedures. User-defined compositions could be saved and reused as new building blocks in other situations (the very nature of component-based). They could also be adapted to cope with a new problem resolution;
- *Analyzing the solution*: often, this approach allows program execution immediately after program creation or simulation within the EUD environment to check whether it does what it should.

A system implementing this technique is the Exercise Creator environment offered to physiotherapists in the frame of the TagTrainer project (Tetteroo, 2017). This environment presents a simple GUI through which physiotherapists can create personalized exercises that foresee manipulating RFID-tagged objects on a tabletop surface. Creating an exercise means positioning 'action blocks' (e.g., 'place object', 'lift object', etc.) on a timeline so that the patient can then execute it on the surface, obtaining audio-visual feedback. Exercise adaptation can also be carried out in TagTrainer Exercise Creator by parameter setting or altering exercise structure, still using direct manipulation on the GUI.

4.2. Component-based and conversational interface (CB-CI)

If we consider the technique that combines the component-based paradigm with conversation-based interaction, the CT model plays the following role:

- *Defining the problem*: components are associated with keywords representing concepts of the application domain. These keywords must be known to the user; furthermore, the user must understand what the system is asking. In other words, the dialogue should be based on the user's language. In this case, the abstraction skill is not needed, but the cognitive effort of modeling the problem appropriately is on the designer's shoulders;
- *Solving the problem*: structuring the dialogue in order to identify all keywords representing the components of the solution is of paramount importance. The user-system dialogue may be already structured in sub-problems, or the decomposition load could be on the user's shoulders, who has to make reasonable requests to the machine to solve the problem. Dialogues are structured to elicit the components that must be connected in sequence, and some keywords may represent repetitions of actions. Based on the implementation of the conversation, connections among components could be automatically generated throughout the dialogue, or the user must know how to define the correct sequence of components, possessing, in this case, an adequate algorithm design skill. User-defined compositions could be associated with new keywords, thus becoming components that can be re-used in further dialogues. If this feature is implemented and suggested in the EUD environment, this could foster the acquisition of the generalization skill;
- *Analyzing the solution*: this approach could allow the visualization of the resulting program in a textual or graphical interface to analyze its correctness and possibly activate its execution, but such description should be suitable to the user's background and education.

For example, the system CAPIRCI (Beschi et al., 2019) offers a chat-based interface to help non-technical users create programs for a collaborative robot. Pick-and-place tasks can be defined by replying to the system questions concerning which object to pick, where to put the object, which kind of manipulation perform on the object, and so on. At the end of the dialogue through the chat, the user may access a graphic interface to verify and modify the robot program. Two user experiments demonstrated how the users appreciated an approach that starts with a natural and intuitive interaction and how this approach can foster gradual learning of robot programming (Fogli et al., 2022).

4.3. Rule-based and direct manipulation (RB-DM)

The analysis of the technique that combines the rule-based paradigm with direct manipulation, concerning the CT model, leads to the following considerations:

- *Defining the problem*: the problem is modeled in terms of actions to be activated when events and/or conditions occur. The user identifies events, conditions and actions, abstracting away from everything else (e.g., how to verify events or perform actions). Events, conditions, and actions are usually presented in available lists: the user must understand their meaning in terms of possible elements for rule composition and be able to distinguish events from conditions and from actions;
- *Solving the problem*: solving the problem means in this context defining new behaviors of a system in terms of IF-THEN rules; thus, the decomposition of the problem consists of the definition of a *trigger_expression* and an *action_expression*: each *trigger_expression* and *action_expression* can be in turn a decomposable expression. Usually, the trigger is specified before the action (although not mandatory). Then, the user can sequentially specify the list of actions to be performed. The GUI of the EUD environment usually presents the user interaction controls to define the event and/or condition first, and then the action(s) by possibly asking configuration parameters if needed; the technique, therefore, allows one to plan a solution with a step-by-step procedure. It could be possible to save user-defined rules and then reuse them as-is or possibly adapt them to solve new problems;

- *Analyzing the solution*: it is usually possible to simulate the execution of a rule by a simple click on it to evaluate whether it is correct or not.

As an example, the work by Manca et al. (2021) applied the rule-based approach to an industrial scenario in the paper sector by providing a visual rule editor that allows domain experts to personalize the behavior of a factory according to events and situations occurring in it. The user can create rules to monitor the production line, manage emergency situations, or perform data analysis and reporting. The rule editor makes triggers and actions available in specific classes (lists of elements) organized according to a logical hierarchy; the user can select them by expanding the relevant classes until the leaves of the hierarchy through simple point-and-click operations. Rules can be saved with a user-defined name and shared with other users. They can be finally activated in the rule editor using the event simulator and checking whether the actions displayed in the action simulator are those expected.

4.4. Rule-based and conversational interface (RB-CI)

The technique RB-CI combines the rule-based paradigm with conversational interaction; with respect to the CT model, it behaves as follows:

- *Defining the problem*: the name of events, conditions, and actions suggested by the system must represent concrete events, conditions, and actions for the user. These names must correspond to the names adopted in that specific work domain and be familiar to the user; alternatively, users could describe their request in their language, and the system should be able to infer the events, conditions, and actions that make up a *trigger_expression* and an *action_expression*;
- *Solving the problem*: when the dialogue is guided by the system (e.g., a chatbot), it will provide users with options among which making their choices. Thus, decomposition is already predefined in the dialogue structure. Alternatively, the system can extract the relevant entities from the dialogue leaving the user free to express their requests, provided that requests have an IF-THEN structure. In the latter case, the user must be able to decompose the problem into a *trigger_expression* and an *action_expression*. When the system guides the user throughout event/condition/action elicitation, it automatically allows one to plan a solution with a step-by-step procedure; otherwise, when the system is in charge of inferring the rules from users' utterances, the user might be required to structure the request according to an IF-THEN structure. A defined rule can be assigned a name and be reused in future dialogues or possibly adapted to solve a new problem;
- *Analyzing the solution*: rules can be activated by asking for their execution, thus permitting to check their correctness.

We provide two examples of EUD environments implementing the RB-CI technique in two different ways. The first is ParlAmI (Stefanidi et al., 2019), a chatbot that helps the user define IF-THEN rules controlling a smart environment. Abstraction is not required to the user, who can speak in natural language, referring to daily events and actions of smart things available in the environment. Rules can be built step-by-step or pronounced in a single sentence. In the step-by-step approach, the chatbot repeats the last acquired message before asking a new question, thus facilitating decomposition, algorithm design, and evaluation. Otherwise, when the user provides a complex rule in a unique message, the system can decompose it and find the event and action(s) to be combined in a rule. A name for the rule is requested at the beginning or at the end of the creation process to promote its reusability.

The second example is HeyTAP, a system able to suggest IF-THEN rules to customize the behavior of an IoT environment based on a request made by the user (Corno et al., 2020, 2021). It exploits a conversational agent to which the user can communicate the customization intention by specifying a request at a high level, such as 'I would like to decrease the temperature of my places'. The system implements a semantic recommendation process to suggest the most suitable rules that satisfy the user's intention, that are supported by the connected entities, and that reflect the user's long-term preferences. If the user cannot find a rule among those suggested, collaboration starts with the system to refine the recommendations. Thus, in this case, users are not required to do any abstraction, decomposition activity, or algorithm design, even though they cannot reuse the selected rule and test it.

4.5. Programming-by-demonstration and direct manipulation (PbD-DM)

The analysis of the PbD-DM technique, which combines programming-by-demonstration with direct manipulation, with respect to the CT model leads to the following considerations:

- *Defining the problem*: the data and actions that users can identify to create new programs must be relevant to them. The user should recognize visual representations of data and actions or physical device manipulations as relevant in that specific work domain. Direct manipulation in a GUI or physical manipulation of a hardware device may favor abstraction;
- *Solving the problem*: The user must be able to divide the activity to be performed into steps (actions) to be recorded, finding their representation in the visual or physical manipulation of the interface; otherwise, decomposition may become difficult for the user. The user must show the machine step-by-step what to do, so the technique requires that the user can plan a step-by-step procedure and perform the correct manipulation sequence. The main challenge of PbD is generalization: the system should generalize automatically but in this case it needs several examples; otherwise, the user must modify the recorded program and be able to change its parameter values to generalize it or adapt it to solve a new problem. Therefore, generalization could be less or more difficult for the user, depending on the capability of the system to perform this activity;
- *Analyzing the solution*: the possibility of reviewing the registration through execution in a graphical interface or a physical device may favor this process.

As mentioned before, PbD is often used in EUP of robotic tasks. Physical manipulation of the robot arm is repeated for a series of example movements under different conditions to make the robot infer a generalized task model. In this way, the robot will be able to reproduce the task in each specific situation. To facilitate task definition by avoiding that the user provides a huge number of example movements, Alexandrova et al. (2014) proposed an approach that combines robot physical manipulation with direct manipulation in a GUI: in this way, the user can demonstrate the task just once, and then visualize the actions learned and customize them through a graphical interface. This approach favors generalization and evaluation. A critical point is instructing the user about the list of available robot commands and their visualization in the system; this requires operating at a level of

abstraction comprehensible for the user both in terms of single commands (abstraction skill) and of their meaning concerning a complex task (problem decomposition skill). Algorithm design is instead facilitated by the implemented approach since, during the demonstration, the user interacts with the system through a simple state-based dialogue controlled by the system (Alexandrova et al., 2014).

4.6. Programming-by-demonstration and conversational interface (PbD-CI)

Finally, by combining the programming-by-demonstration paradigm with conversational interaction, one may obtain PbD-CI that implies what follows:

- *Defining the problem*: description of actions and data is carried out in natural language, using terms that should be familiar to the user;
- *Solving the problem*: the user is required to describe the activity to be performed by specifying the steps (actions) to be recorded. In particular, the user must describe in natural language what the machine must do step-by-step, so the technique requires that the user can plan a step-by-step procedure. Also, in this case, the system should generalize automatically, or the user may be required to generalize the resulting program. Generalization could be less or more difficult for the user, depending on the capability of the system to perform this activity;
- *Analyzing the solution*: the actions extracted from the user's description can be told by the system or represented in a visual form to allow checking correctness. The possibility of reviewing the registration in another type of interface may favor the acquisition of the evaluation skill.

PbD usually requires a visual or physical manipulation of an interface through which the user demonstrates an activity; however, generalization requires several activity executions. To address this issue, such manipulation can be combined with a natural language description through which the user may provide instructions that allow to detect the user's action reliably and on which object the user acted. This approach is implemented in APPINITE (Li et al., 2018), a EUD environment for creating automation tasks for smart devices (phones, wearables, appliances, and speakers). Problem decomposition and algorithm design are favored by a step-by-step dialogue guided by the system. APPINITE allows the user to specify data descriptions through a multi-initiative conversation, try out different instructions, and review the demonstrated actions without executing them (evaluation). To have good coverage for words and expressions used by the users, and thus to favor abstraction, a big dataset is, however, needed to train the parser.

5. EUDability: definition and dimensions

As outlined in Section 1, the design and assessment of End-User Development applications usually consider the usability construct only. Considering the outcomes of Section 4 describing the interplay between EUD techniques and Computational Thinking, we believe that the usability construct is not enough to guide the design and, later, the assessment of EUD systems. Indeed, the definition of *usability* that is commonly accepted by the scientific community is the one provided by ISO in its standard 9241-11, first published in 1998 and updated several times until recently in 2018 (ISO, 2018): "The degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use". Therefore, according to this standard, *effectiveness*, *efficiency*, and *satisfaction* are the three main dimensions that characterize usability. The above definition stems from a long-lasting process, and one of the main experts on the matter is Jakob Nielsen. He identified five basic dimensions that are required at a minimum to consider a product or system usable (Nielsen, 1994): *learnability*, *efficiency*, *memorability*, *robustness*, and *satisfaction*.

The usability construct can be applied to a variety of products and systems. Its definition considers the existence of different kinds of users operating in different contexts to achieve different goals. However, End-User Development, applied in specific work scenarios, presents some more peculiar aspects that need to be considered. There is a need to capture the dual aspect discussed in Section 1, which relates to the capabilities of the system to support EUD activities on one side and foster the user's Computational Thinking on the other side.

In line with the five CT skills that must be activated and applied in each stage of a EUD problem, we identity five dimensions:

- *Concreteness*: this dimension refers to the capability of a EUD environment of presenting concepts and requests in a concrete way, without requiring the user highly-developed abstraction skills;
- *Modularity*: this dimension regards the availability in a EUD environment of different elements, blocks, modules or similar things, which help end-user developers decompose a problem and identify the pieces that may compose its solution;
- *Structuredness*: when a EUD environment supports the structuring of a solution in a step-by-step process and facilitates the connections between the input and output of the different steps, it can be considered as having the structuredness property;
- *Reusability*: when the outcome of a EUD activity can be reused in other situations and possibly shared among different end-user developers, the EUD environment possesses the reusability property;
- *Testability*: this dimension refers to the capability of testing the outcome of the EUD activity within the EUD environment.

The relationship between these dimensions and the CT skills is depicted in Fig. 5.

Given the above dimensions, we may provide the following definition: *EUDability is the degree of concreteness, modularity, structuredness, reusability, and testability fostered by a EUD environment designed for specified end-user developers, with a specified goal to be pursued in a specified context.*

By referring to *specified end-user developers*, this definition would highlight that the human workers called on to perform EUD activities may vary in terms of their CT skills, which must be accommodated and possibly improved by the EUD techniques and their implementation. Thus, like usability is not a property of an interactive system *per se*, but depends on the profile of the target users, similarly, EUDability depends on the characteristics and roles of end-user developers. The *goal* of the EUD activity is also crucial to evaluate the degree of EUDability of a system. We recall that the main goals of EUD can be: (i) modification (adaptation) of an existing artifact; (ii) extension of an artifact with new features; or (iii) creation of a new artifact. Reaching these goals could therefore be less or more difficult, and again the system EUDability can help address such difficulty. Finally, the *context* in which the EUD activity occurs is related to a specific work scenario. Thus, the type of work and expertise it requires can make a EUD environment more or less suitable for that work scenario.

In summary, a EUD environment can implement one or more EUD techniques that can foster EUDability in different ways, facilitating end-user developers' work and helping them gradually

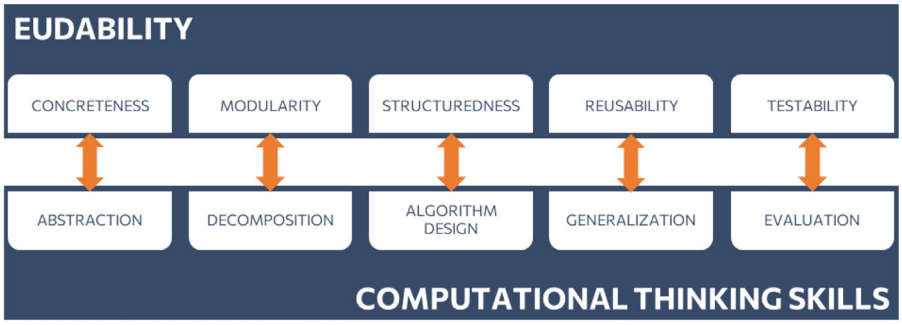

Fig. 5. Each one of the five EUDability dimensions is directly connected with one of the five Computational Thinking skills.

acquire the CT skills necessary to cope with more and more complex problems. The six EUD techniques discussed in the previous section provide some examples in this direction.

6. Conclusion

This paper aimed to explore the interplay between End-User Development and Computational Thinking to identify new design practices for the future of workplaces. The main contributions of the paper consist of: (i) an original characterization of the main EUD techniques adopted in specific work scenarios, which combines the EUD paradigm with the interaction style adopted in the EUD environment; (ii) a comprehensive model of CT that fosters the crossover with EUD techniques; (iii) the definition of the EUDability construct, based on the previous two contributions, to be considered in the design and assessment of EUD environments, which complements the well-known usability construct.

Several limitations affect the research carried out so far. First of all, we selected for our investigation only the three most used EUD paradigms and only two interaction styles; other paradigms and styles could be included in the analysis to obtain a more comprehensive characterization of EUD techniques. Furthermore, we did not report on the application of paradigm combinations; however, we do not exclude the fact that two paradigms, such as the rule-based and component-based ones, could be integrated in the same EUD environment; the considerations made in Section 5 would remain valid also in these cases, but a deeper investigation is needed. Last but not least, we believe that this work is only the first step of a long-term journey. The plan for future research includes: (i) the definition of a methodology for assessing CT skills of end-user developers; (ii) the specification of a methodology for designing EUD environments, which takes into account EUDability dimensions, possibly formulating a set of EUDability heuristics, similarly to what happened for usability; (iii) the definition of a methodology for assessing EUDability, which, contrarily to usability evaluation, considers the dual aspect related to the difficulty of using a EUD tool and the level of end-user developers' CT skills.

CRediT authorship contribution statement

Barbara Rita Barricelli: Conceptualization, Methodology, Writing. **Daniela Fogli:** Conceptualization, Methodology, Writing. **Angela Locoro:** Conceptualization, Methodology, Writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Abdelnour-Nocera, J., Barricelli, B.R., Lopes, A., Campos, P., Clemmensen, T., 2015. Preface. IFIP Adv. Inf. Commun. Technol. 468, V.

Alexandrova, S., Cakmak, M., Hsiao, K., Takayama, L., 2014. Robot programming by demonstration with interactive action visualizations. In: Robotics: Science and Systems. pp. 48–56. http://dx.doi.org/10.15607/RSS.2014.X.048.

Ardito, C., Buono, P., Desolda, G., Matera, M., 2018. From smart objects to smart experiences: An end-user development approach. Int. J. Hum.-Comput. Stud. 114, 51–68. http://dx.doi.org/10.1016/j.ijhcs.2017.12.002.

Asunis, L., Frau, V., Macis, R., Pireddu, C., Spano, L.D., 2021. PAC-Bot: Writing text messages for developing point-and-click games. In: Fogli, D., Tetteroo, D., Barricelli, B.R., Borsci, S., Markopoulos, P., Papadopoulos, G.A. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 213–221. http://dx.doi.org/10.1007/978-3-030-79840-6_15.

Barakova, E., Gillesen, J., Huskens, B., Lourens, T., 2013. End-user programming architecture facilitates the uptake of robots in social therapies. Robot. Auton. Syst. 61 (7), 704–713. http://dx.doi.org/10.1016/j.robot.2012.08.001.

Barr, V., Stephenson, C., 2011a. Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? ACM Inroads 2 (1), 48–54. http://dx.doi.org/10.1145/1929887.1929905.

Barr, V., Stephenson, C., 2011b. Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? ACM Inroads 2 (1), 48–54. http://dx.doi.org/10.1145/1929887.1929905.

Barricelli, B.R., Cassano, F., Fogli, D., Piccinno, A., 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. J. Syst. Softw. 149, 101–137. http://dx.doi.org/10.1016/j.jss.2018.11.041.

Barricelli, B.R., Fischer, G., Fogli, D., Mørch, A., Piccinno, A., Valtolina, S., 2016. Cultures of participation in the digital age: From "have to" to "want to" participate. In: NordiCHI '16: Proceedings of the 9th Nordic Conference on Human-Computer Interaction. NordiCHI '16, Association for Computing Machinery, New York, NY, USA, pp. 1–3. http://dx.doi.org/10.1145/2971485.2987668.

Barricelli, B.R., Valtolina, S., 2017. A visual language and interactive system for end-user development of internet of things ecosystems. J. Vis. Lang. Comput. 40, 1–19. http://dx.doi.org/10.1016/j.jvlc.2017.01.004.

Barricelli, B.R., Valtolina, S., Gadia, D., Marzullo, M., Piazzi, C., Garzulino, A., 2015. Participatory action design research in archaeological context. In: Abdelnour Nocera, J., Barricelli, B.R., Lopes, A., Campos, P., Clemmensen, T. (Eds.), Human Work Interaction Design. Work Analysis and Interaction Design Methods for Pervasive and Smart Workplaces. Springer International Publishing, Cham, pp. 192–211. http://dx.doi.org/10.1007/978-3-319-27048-7_14.

Beschi, S., Fogli, D., Tampalini, F., 2019. CAPIRCI: A multi-modal system for collaborative robot programming. In: Malizia, A., Valtolina, S., Morch, A., Serrano, A., Stratton, A. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 51–66. http://dx.doi.org/10.1007/978-3-030-24781-2_4.

Brennan, K., Resnick, M., 2012. New frameworks for studying and assessing the development of computational thinking. In: Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada, Vol. 1, p. 25.

Chkroun, M., Azaria, A., 2019. Lia: A virtual assistant that can be taught new commands by speech. Int. J. Hum.–Comput. Interact. 35 (17), 1596–1607. http://dx.doi.org/10.1080/10447318.2018.1557972.

Corno, F., De Russis, L., Roffarello, A.M., 2020. HeyTAP: Bridging the gaps between users' needs and technology in IF-THEN rules via conversation. In: Proceedings of the International Conference on Advanced Visual Interfaces. AVI '20, Association for Computing Machinery, New York, NY, USA, pp. 1–9. http://dx.doi.org/10.1145/3399715.3399905.

Corno, F., De Russis, L., Roffarello, A.M., 2021. From users' intentions to IF-THEN rules in the internet of things. ACM Trans. Inf. Syst. 39 (4), http://dx.doi.org/10.1145/3447264.

Costabile, M.F., Fogli, D., Marcante, A., Mussio, P., Provenza, L.P., Piccinno, A., 2008. Designing customized and tailorable visual interactive systems. Int. J. Softw. Eng. Knowl. Eng. 18 (3), 305–325. http://dx.doi.org/10.1142/S0218194008003702.

Dey, A.K., Hamid, R., Beckmann, C., Li, I., Hsu, D., 2004. A CAPpella: Programming by demonstration of context-aware applications. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '04, Association for Computing Machinery, New York, NY, USA, pp. 33–40. http://dx.doi.org/10.1145/985692.985697.

DiSessa, A.A., 2018. Computational literacy and "the big picture" concerning computers in mathematics education. Math. Think. Learn. 20 (1), 3–31. http://dx.doi.org/10.1080/10986065.2018.1403544.

Fanni, F.A., Senis, M., Tola, A., Murru, F., Romoli, M., Spano, L.D., Blecic, I., Trunfio, G.A., 2019. PAC-pac: End user development of immersive point and click games. In: Malizia, A., Valtolina, S., Morch, A., Serrano, A., Stratton, A. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 225–229. http://dx.doi.org/10.1007/978-3-030-24781-2_20.

Fischer, G., 2002. Beyond "couch potatoes": From consumers to designers and active contributors. First Monday 7 (12), http://dx.doi.org/10.5210/fm.v7i12.1010.

Fischer, G., Fogli, D., Piccinno, A., 2017. Revisiting and broadening the meta-design framework for end-user development. In: Paternò, F., Wulf, V. (Eds.), New Perspectives in End-User Development. Springer International Publishing, Cham, pp. 61–97. http://dx.doi.org/10.1007/978-3-319-60291-2_4.

Fischer, G., Giaccardi, E., 2006. Meta-design: A framework for the future of end-user development. In: Lieberman, H., Paternò, F., Wulf, V. (Eds.), End User Development. Springer Netherlands, Dordrecht, pp. 427–457. http://dx.doi.org/10.1007/1-4020-5386-X_19.

Fletcher, G.H.L., Lu, J.J., 2009. Human computing skills: Rethinking the K-12 experience. Commun. ACM 52 (2), 23–25. http://dx.doi.org/10.1145/1461928.1461938.

Fogli, D., Gargioni, L., Guida, G., Tampalini, F., 2022. A hybrid approach to user-oriented programming of collaborative robots. Robot. Comput.-Integr. Manuf. 73, 102234. http://dx.doi.org/10.1016/j.rcim.2021.102234.

Ghiani, G., Manca, M., Paternò, F., Santoro, C., 2017. Personalization of context-dependent applications through trigger-action rules. ACM Trans. Comput.-Hum. Interact. 24 (2), http://dx.doi.org/10.1145/3057861.

Grover, S., Pea, R., 2013. Computational thinking in K–12: A review of the state of the field. Educ. Res. 42 (1), 38–43. http://dx.doi.org/10.3102/0013189X12463051.

Grudin, J., 1991. CSCW: The convergence of two development contexts. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '91, Association for Computing Machinery, New York, NY, USA, pp. 91–97. http://dx.doi.org/10.1145/108844.108858.

Hamabe, T., Goto, H., Miura, J., 2015. A programming by demonstration system for human-robot collaborative assembly tasks. In: 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO). pp. 1195–1201. http://dx.doi.org/10.1109/ROBIO.2015.7418934.

Hang, F., Zhao, L., 2015. Supporting end-user service composition: A systematic review of current activities and tools. In: 2015 IEEE International Conference on Web Services. pp. 479–486. http://dx.doi.org/10.1109/ICWS.2015.70.

Hsu, T.-C., Chang, S.-C., Hung, Y.-T., 2018. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. Comput. Educ. 126, 296–310. http://dx.doi.org/10.1016/j.compedu.2018.07.004.

Huang, T.-H.K., Azaria, A., Romero, O.J., Bigham, J.P., 2019. InstructableCrowd: Creating IF-THEN rules for smartphones via conversations with the crowd. Hum. Comput. 6, 113–146. http://dx.doi.org/10.15346/hc.v6i1.7.

Huang, J., Cakmak, M., 2017. Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI). pp. 453–462.

ISO, 2018. Ergonomics of Human-System Interaction — Part 11: Usability: Definitions and Concepts. Standard, International Organization for Standardization, Geneva, CH.

Kaasinen, E., Aromaa, S., Väätänen, A., Mäkelä, V., Hakulinen, J., Keskinen, T., Elo, J., Siltanen, S., Rauhala, V., Aaltonen, I., Hella, J., Honkamaa, P., Leppä, M., Niemelä, A., Parviainen, J., Turunen, M., Törnqvist, J., Valtonen, J., Woodward, C., 2019. Mobile service technician 4.0: Knowledge-sharing solutions for industrial field maintenance. Interact. Des. Archit.(S) (ISSN: 1826-9745) (38), 6–27.

Kusmin, K.-L., Tammets, K., Ley, T., 2018. University-industry interoperability framework for developing the future competences of industry 4.0. Interact. Des. Archit.(S) 38, 28–45.

Li, T.J.-J., Azaria, A., Myers, B.A., 2017. SUGILITE: Creating multimodal smartphone automation by demonstration. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. CHI '17, Association for Computing Machinery, New York, NY, USA, pp. 6038–6049. http://dx.doi.org/10.1145/3025453.3025483.

Li, T.J.-J., Labutov, I., Li, X.N., Zhang, X., Shi, W., Ding, W., Mitchell, T.M., Myers, B.A., 2018. APPINITE: A multi-modal interface for specifying data descriptions in programming by demonstration using natural language instructions. In: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 105–114. http://dx.doi.org/10.1109/VLHCC.2018.8506506.

Lieberman, H., Paternò, F., Wulf, V., 2006. End User Development (Human-Computer Interaction Series). Springer-Verlag, Berlin, Heidelberg, http://dx.doi.org/10.1007/1-4020-5386-X.

Maceli, M.G., 2017. Tools of the trade: A survey of technologies in end-user development literature. In: Barbosa, S., Markopoulos, P., Paternò, F., Stumpf, S., Valtolina, S. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 49–65. http://dx.doi.org/10.1007/978-3-319-58735-6_4.

Manca, M., Paternò, F., Santoro, C., 2021. Personalization in a paper factory. In: Fogli, D., Tetteroo, D., Barricelli, B.R., Borsci, S., Markopoulos, P., Papadopoulos, G.A. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 102–118. http://dx.doi.org/10.1007/978-3-030-79840-6_7.

Meister, J.C., Brown, R.H., 2020. 21 HR jobs of the future. Harv. Bus. Rev..

Merisotis, J., 2020. Human Work in the Age of Smart Machines. RosettaBooks, New York, USA.

Mørch, A., Hansen, Å., Hege-René, 2006. Super users and local developers: The organization of end user development in an accounting company. J. Organ. End User Comput. 18 (4), 1–21. http://dx.doi.org/10.4018/joeuc.2006100101.

Moreno-León, J., Robles, G., Román-González, M., 2015. Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. RED. Rev. Educ. Dist. (46), 1–23.

Nardi, B.A., Miller, J.R., 1990. An ethnographic study of distributed problem solving in spreadsheet development. In: Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work. CSCW '90, Association for Computing Machinery, New York, NY, USA, pp. 197–208. http://dx.doi.org/10.1145/99332.99355.

Nielsen, J., 1994. Usability Engineering. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Palts, T., Pedaste, M., 2020. A model for developing computational thinking skills. Inform. Educ. 19 (1), 113–128. http://dx.doi.org/10.15388/infedu.2020.06.

Parsa, S., Saadat, M., 2021. Human-robot collaboration disassembly planning for end-of-life product disassembly process. Robot. Comput.-Integr. Manuf. 71, 102170. http://dx.doi.org/10.1016/j.rcim.2021.102170.

Paternò, F., 2013. End user development: Survey of an emerging field for empowering people. ISRN Softw. Eng. http://dx.doi.org/10.1155/2013/532659.

Paternò, F., Wulf, V. (Eds.), 2017. New Perspectives in End-User Development. Springer, Cham, http://dx.doi.org/10.1007/978-3-319-60291-2.

Pollak, M., Ebner, M., 2019. The missing link to computational thinking. Future Internet 11 (12), 263. http://dx.doi.org/10.3390/fi11120263.

Ponce, V., Abdulrazak, B., 2022. Context-aware end-user development review. Appl. Sci. 12 (1), http://dx.doi.org/10.3390/app12010479.

PwC, 2018. Workforce of the future: The competing forces shaping 2030. URL: https://www.pwc.com/gx/en/services/people-organisation/workforce-of-the-future/workforce-of-the-future-the-competing-forces-shaping-2030-pwc.pdf.

Repenning, A., Ioannidou, A., 2006. What makes end-user development tick? 13 design guidelines. In: End User Development. Springer Netherlands, Dordrecht, pp. 51–85. http://dx.doi.org/10.1007/1-4020-5386-X_4.

Román-González, M., Pérez-González, J.-C., Jiménez-Fernández, C., 2017. Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. Comput. Hum. Behav. 72, 678–691. http://dx.doi.org/10.1016/j.chb.2016.08.047.

Schou, C., Andersen, R.S., Chrysostomou, D., Bgh, S., Madsen, O., 2018. Skill-based instruction of collaborative robots in industrial settings. Robot. Comput.-Integr. Manuf. 53, 72–80. http://dx.doi.org/10.1016/j.rcim.2018.03.008.

Selby, C., Woollard, J., 2013. Computational Thinking: the Developing Definition. Technical Report, University of Southampton (E-prints).

Stefanidi, E., Foukarakis, M., Arampatzis, D., Korozi, M., Leonidis, A., Antona, M., 2019. ParlAmI: A multimodal approach for programming intelligent environments. Technologies 7 (1), 1–31. http://dx.doi.org/10.3390/technologies7010011.

Tetteroo, D., 2017. Tagtrainer: End-user adaptable technology for physical rehabilitation. In: Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare. PervasiveHealth '17, Association for Computing Machinery, New York, NY, USA, pp. 452–454. http://dx.doi.org/10.1145/3154862.3154901.

Tetteroo, D., Markopoulos, P., 2015. A review of research methods in end user development. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 58–75. http://dx.doi.org/10.1007/978-3-319-18425-8_5.

Tsai, M.-J., Liang, J.-C., Hsu, C.-Y., 2021. The computational thinking scale for computer literacy education. J. Educ. Comput. Res. 59 (4), 579–602. http://dx.doi.org/10.1177/0735633120972356.

Turchi, T., Fogli, D., Malizia, A., 2019. Fostering computational thinking through collaborative game-based learning. Multimedia Tools Appl. 78 (10), 13649–13673. http://dx.doi.org/10.1007/s11042-019-7229-9.

Ur, B., McManus, E., Pak Yong Ho, M., Littman, M.L., 2014. Practical trigger-action programming in the smart home. In: Proc. of the SIGCHI Conference on Human Factors in Computing Systems. CHI '14, Association for Computing Machinery, New York, NY, USA, pp. 803–812. http://dx.doi.org/10.1145/2556288.2557420.

Valtolina, S., Barricelli, B.R., Fogli, D., Colosio, S., Testa, C., 2017. Public staff empowerment in e-government: A human work interaction design approach. In: Barbosa, S., Markopoulos, P., Paternò, F., Stumpf, S., Valtolina, S. (Eds.), End-User Development. Springer International Publishing, Cham, pp. 119–134. http://dx.doi.org/10.1007/978-3-319-58735-6_9.

Wang, C., Shen, J., Chao, J., 2021. Integrating computational thinking in STEM education: A literature review. Int. J. Sci. Math. Educ. 1–24. http://dx.doi.org/10.1007/s10763-021-10227-5.

Weintrop, D., Shepherd, D.C., Francis, P., Franklin, D., 2017. Blockly goes to work: Block-based programming for industrial robots. In: 2017 IEEE Blocks and beyond Workshop (B B). pp. 29–36. http://dx.doi.org/10.1109/BLOCKS.2017.8120406.

Whiley, K., Blackwell, A.F., 2001. Visual programming in the wild: A survey of LabVIEW programmers. J. Vis. Lang. Comput. 12 (4), 435–472. http://dx.doi.org/10.1006/jvlc.2000.0198.

Wing, J.M., 2006. Computational thinking. Commun. ACM 49 (3), 33–35. http://dx.doi.org/10.1145/1118178.1118215.

Wing, J., 2011. Research notebook: Computational thinking—What and why. Link Mag. 6, 20–23, https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why.

World Economic Forum, 2020. The future of jobs report 2020. URL: https://www.weforum.org/reports/the-future-of-jobs-report-2020.

Barbara Rita Barricelli is Associate Professor at the Department of Information Engineering, University of Brescia, Italy. Her research interests are Human–Computer Interaction, End-User Development, Computer Semiotics and Semiotic Engineering, and Participatory Design. She is Chair of the IFIP Working Group TC13.6 on Human Work Interaction Design.

Daniela Fogli is Full Professor at the Department of Information Engineering, University of Brescia, Italy. Her research interests include methods for designing complex interactive systems, meta-design, end-user development, web usability and accessibility, decision support systems. She is serving as senior associate editor for the Decision Support Systems journal, and she is chair of the steering committee of the International Symposium on End-User Development (IS-EUD).

Angela Locoro is Assistant Professor (Tenure Track) at the Department of Theoretical and Applied Sciences University of Insubria, Varese, Italy. Her research interests include Data Visualization, Visual Information Literacy, Human–Computer Interaction, Knowledge Management and Information Processing. She served as committee member, chair and track chair in many international conferences about Human–Computer Interaction and Information Systems.