DESIGNING A DIGITAL LEARNING ENVIRONMENT FOR COMPUTATIONAL THINKING: THE FOUR PILLARS OF <COLETTE/>

Rebecca S. Stäter[1], Matthias Ludwig[1]

[1] *Goethe University Frankfurt; staeter@math.uni-frankfurt.de, ludwig@math.uni-frankfurt.de*

Computational Thinking is an essential skillset for participating in and shaping our increasingly digital world, and it needs to be explicitly taught in schools. However, teaching material for Computational Thinking is sparse, especially for contexts outside of Computer Science. In this article, we present our approach to designing the digital learning environment <colette/>, which facilitates teaching Computational Thinking in schools. We discuss the design rationale behind the four core pillars (app, web portal, didactic content, and teacher training), and illustrate how they interact with each other. Communicating our vision of <colette/> will be useful for conceptualizing, implementing, and comparing similar digital learning environments.

Keywords: Computational Thinking, Learning Environment, Mobile Learning

INTRODUCTION

Computational Thinking (CT) is a term coined by Jeanette Wing in 2006, conceptually defined as:

> Computational thinking is a fundamental skill for everyone, not just for computer scientists. […] Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science. (Wing, 2006, p.33)

Since then, several researchers have created more explicit, sometimes incompatible, definitions of CT (Weintrop et al. (2016), Fraillon et al. (2020), and see Bocconi et al. 2016 and 2022 for a more detailed overview). According to Bocconi et al (2016), who synthesized multiple definitions to find a common ground, CT comprises six core skills: Algorithmic Thinking, Debugging, Abstraction, Decomposition, Generalization and Automation.

Abstraction is the skill applied to construct a fitting, mental model for a real-life object by discarding unnecessary details. Sometimes it might be helpful to *decompose* the given problem into smaller problems, solving the smaller ones and combining these solutions to a solution of the firstly given problem. The skill of writing down the way to a solution by using clearly defined steps such that human beings as well as machines can understand and follow them is called *Algorithmic Thinking*. Writing down algorithms also involves mentally working through them step by step to see where a problem might occur as well as how the algorithm can be designed better or more efficiently, which is called *Debugging*. Exploiting the fact that machines can perform repetitive tasks very well is the concept of *Automation*. After solving a problem, one may ask if the solution can be *generalized* to a larger set of similar

Note that CT is not necessarily related to Computer Science (CS); at its core CT simply is a skillset for solving problems (Wing, 2006), and can be taught and applied in other scenarios too. Although there is a huge movement to include CT in classrooms, it is still unclear how to proceed with this. Bocconi et al. (2016) suggests that the educator has three options: introducing CT in a new subject, in an already existing subject or as a non-subject related skill but as a cross-curricula theme. It is still unclear how to proceed with this. In practice teachers often have to educate themselves on how to actually introduce CT in their classes. Coding learning environments such as Scratch, code.org, codeHS, pocket code, or MIT app inventor allow teachers to focus on CT in the Computer Science context, but do not offer teaching material for CT for other school subjects. This means, as teachers do not necessarily need coding expertise, some of them are left behind. In fact teachers ask for easy-to-use material as one of the biggest hurdles to teaching CT (Borkulo et al., 2020).

This paper introduces <colette/>, a digital learning environment designed to facilitate the teaching of computational thinking (CT) in a variety of school subjects / contexts. The platform comprises four core pillars: a web portal for task creation, a smartphone app for task completion, educational materials for teachers, and didactic content. By describing these four pillars in detail, we lay out our design rationales and outline how the different components of <colette/> interact to create a seamless and engaging learning experience. We hope sharing the concept of <colette/> can serve as model to inspire the development of digital learning environments, and contribute to the widespread adoption of innovative and effective educational tools.

THE DIGITAL LEARNING ENVIRONMENT <COLETTE/>

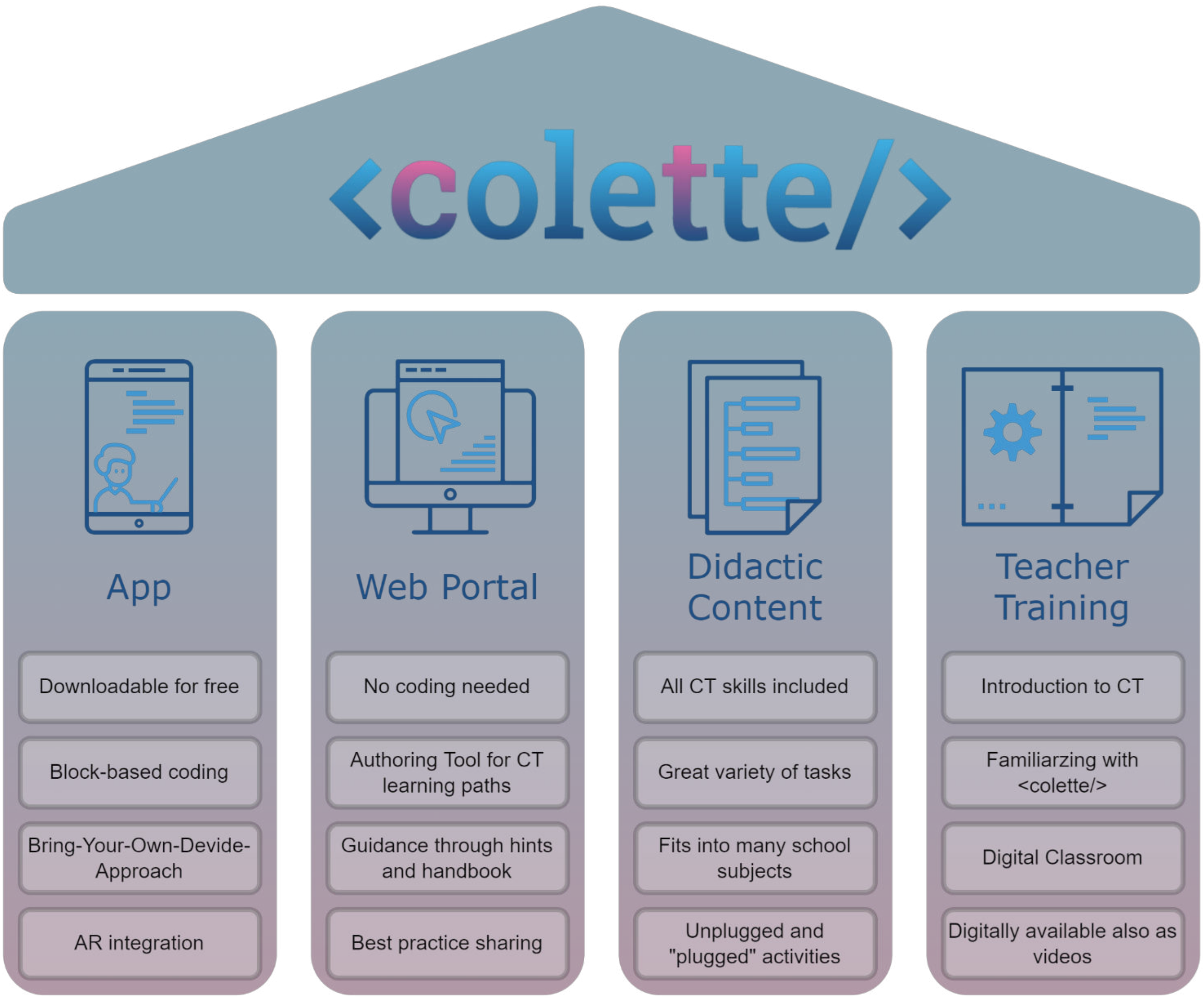
<colette/> is a digital learning environment for CT, conceived by a consortium of six organizations from five countries (Germany, Austria, the Netherlands, France, Slovakia), partially funded as a three-year Erasmus+ project (2020-2023). <colette/> is at the development stage, currently being created and refined by working on the material and further adapting the web portal and the app.

The design of <colette/> consists of four not entirely disjoint pillars (figure 1). The first one is the smartphone app which students use to work on the tasks assigned by the teacher and which will be downloadable for free. The second is the web portal where teachers act as an author and design the learning paths for their students, create new tasks and share best practice. The third one is the didactic content, from which teachers can use the given, adaptable *Task Families* to design their own tasks and the fourth one is teacher education. We designed a teacher training which can be attended in person or can be watched online. Additionally, a handbook will be made available in the near future, guiding teachers to create meaningful learning paths. In Bocconi's latest report (2022) teacher education was mentioned as the biggest challenge educators are facing in the lower secondary schools for introducing CT into their classes, warranting the need for CT-specific teacher education.

THE 1ST PILLAR: THE APP

After having seen the rough project idea, we start with the first pillar by looking at <colette/> in classrooms in action. We developed the app to be compatible with both tablets and smartphones, and it can be downloaded for free on both iOS and Android platforms. By enabling students to use their own devices ("Bring-your-own-device" approach), <colette/> eliminates the need for schools to invest in expensive technology. Not requiring any account creation allows for a simple and streamlined onboarding process.

The learning environment is meant to host various non-coding exercises to provide a comprehensive CT learning experience. However, when coding is involved, a block-based language is used, meaning that coding novices have a low threshold to getting started (Weintrop, 2019).



<colette/>
App
Downloadable for free
Block-based coding
Bring-Your-Own-Devide-Approach
AR integration
Web Portal
No coding needed
Authoring Tool for CT learning paths
Guidance through hints and handbook
Best practice sharing
Didactic Content
All CT skills included
Great variety of tasks
Fits into many school subjects
Unplugged and "plugged" activities
Teacher Training
Introduction to CT
Familiarzing with <colette/>
Digital Classroom
Digitally available also as videos

Figure 1: The four pillars of the learning environment <colette/>: App, web portal, didactic content, and teacher training.

The app offers an AR-view enabling the students to verify the function of their code before submitting it. For example, students can see a 3D structure created by their code and see as a ghost-view the desired structure, so that students can see which part of their code isn't working yet. In another *Task* Family a drone executes their code step-by-step and capture images along the way. In both *Task Families* is the CT skill *Debugging* fostered through the AR view.

THE 2ND PILLAR: THE WEB PORTAL

We have seen that the app offers the students' perspective on <colette/>. The web portal acts as an authoring tool for the educators letting them adapt the available tasks to their liking. It is designed to being easy to use, no coding is needed to create meaningful CT tasks. The didactic content is being designed by the consortium so the educators only have to adapt them for their classrooms. The web portal guides the educator to create those tasks and paths by giving hints along the way facilitating the onboarding for teachers and the creation of learning paths.

Teachers are able to register in the web portal and tasks they created can be made accessible for other educators. This way, best-practices can be shared easily and will make the work with <colette/> even more efficient.

The web portal gives also access to the handbook which is currently being designed and refined. The handbook has two components. The first component is explanatory text and images which can be found in the web portal itself. The second component are videos which will be created and available in all project languages including English and made accessible on our project website. With this guidance and hints in the portal itself we are creating a low threshold learning environment for CT not only for students but also for educators themselves.

THE 3RD PILLAR: THE DIDACTIC CONTENT

The web portal gives access to so called *Task Families* being a general set of tasks (Stäter et al., 2023) all having a certain common core. For example, the common core of the *Task Family Building Cubes* is that all derived tasks ask the student to place cubes on a 3D grid building a shape. Within this *Task Family* different *Assignment Types* are available (figure 2), meaning implementation, analysis, find the error, etc. For each of the *Assignment Types* different *Scenarios* (meaning different structures) can be selected and edited by using the available set of settings (e.g. size, position, height).

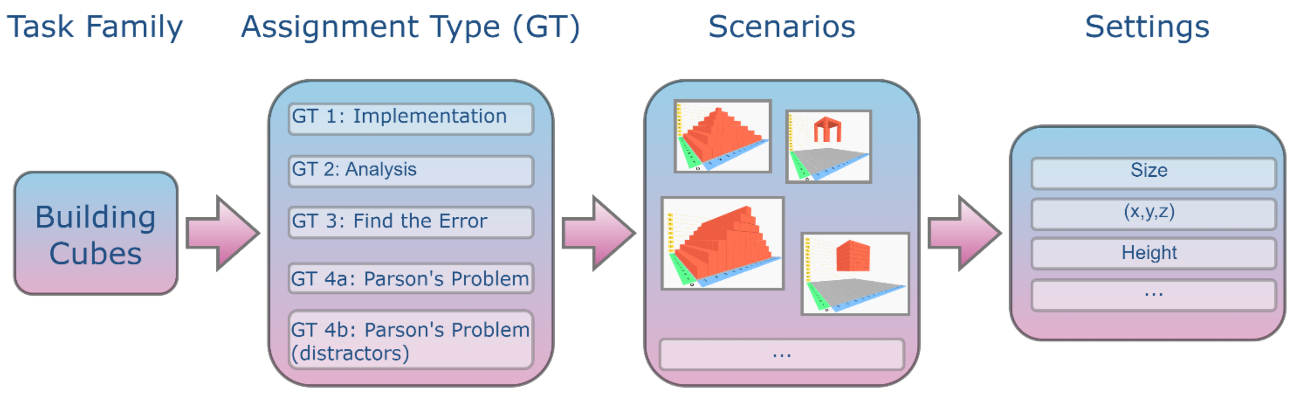


Figure 2: A *Task Family* includes different *Assignment Types* (for coding-based *Task Families* these are the *Generic Task* ideas) and *Scenarios* which can be edited using different *Settings*. In this way from one *Task Family* several different tasks can be derived.

When designing tasks specifically fostering CT skills not only in the context of coding and being usable in the digital learning environment, certain design choices had to be made. In what follows we'll lay out some design considerations for tasks that can be automatically corrected, created, and easily varied. First, we will explain the design concepts for creating CT fostering tasks, then considerations for creating coding tasks that will be automatically validated.

Design Concept of Task Families and Their Connection to CT-skills

For designing the *Task Families,* we used the backward approach: We had the learning goal in mind and created a task that was fitting to offer access to this learning goal (Lyon et al., 2020). Hence, we designed a *Task Family* having the specific CT skill in mind that students should develop working on the given task derived from the specific *Task Family*.

For addressing the skill *Algorithmic Thinking*, e.g., we created among others the Task Family *Drone* where in one of the *Assignment Types* the students code the flying route of a drone which takes pictures on the way of certain spots of a building or structure. Because this *Task Family* with the *Assignment Type* of *Implementation* focuses on the design of the algorithm using block-based language, we assume that this *Task Family* will highly address the skill *Algorithmic Thinking* (Milicic et al., 2020).

Another example for addressing the skill *Algorithmic Thinking* but without coding is based on process diagrams. For this, one can take any real-life process (using a coffee machine, making dinner, …) and create a process diagram out of it, explaining this real-life process to any other human being or machine. *Algorithmic Thinking* is fostered as the students must think about each small step a computer or another human being would have to execute and find one correct order of it. An algorithm, at its core, is nothing else as a set of precise instructions in a given order where one produces from given input the desired output. This *Task Family* is available as an unplugged version now and can be used without having any coding experience in nearly every school subject, even language lessons (Rottenhofer et al., 2021).

When making use of the skill *Abstraction* one takes a real-life situation and takes out any unnecessary details to look at a more abstract version of the real world. For addressing this skill, the *Task Family Graphs* was created. In one of its scenarios the student looks at a real-life birds-view picture of a city and has to abstract it in such a way that in the end one has a proper graph giving only the cities as nods and their distances between them along the vertices (Médova et al., 2022). In this process the student creates from the given picture a more abstract version of it in form of a map and then abstracting it more getting to the graph.

Connecting Task Families and Generic Tasks

Generic Task is a concept introduced by Milicic et al. (2020) in which one can create different standard CS-tasks simply by using three inputs (figure 3): Algorithm, description, and an optional image. Combining these inputs one can create five different output tasks: Implementation, analysis, find the error, Parson's problem (with and without distractors). In <colette/> we use this concept as *Assignment Types* to enrich *Task Families* that are based on coding.

Like mentioned above, <colette/> offers a variety of tasks by using the different *Task Families*. Making use of the concept of *Generic Task*s offers a bigger set of assignment types. The web portal only needs three inputs: Algorithm, description, image. Using these three inputs the portal can create five different assignment types simply by either leaving one of the inputs out (e.g. Implementation, for this the portal doesn't show the algorithm to students) or changing one (e.g. Find the Error, the portal changes something in the code, like a value, and creates, hence, an error). Those different assignment types will be included for all of the coding-based *Task Families* as assignment type.

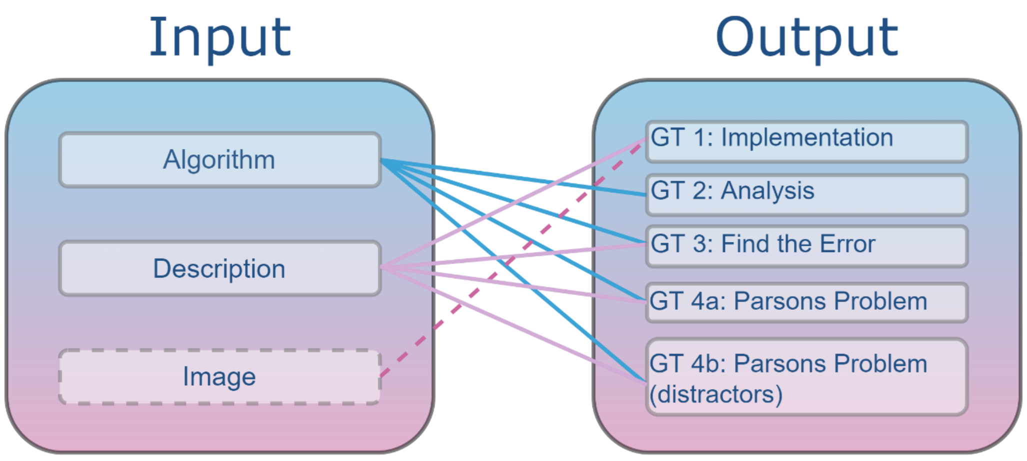


Figure 3: The given input for the task is combined to derive different output tasks. For example, taking the input "Algorithm" and "Description" you can design a "Find the Error" task.

THE 4TH PILLAR: THE TEACHER TRAINING

In the first three pillars, we described the structure of the technology behind <colette/> and the didactic material being used in lessons. <colette/> does not only offer this technology and didactic material but also educates teachers on how to use this technology in a meaningful way to integrate CT in their classrooms. As mentioned previously/above, the lack of adequately trained teachers was found to be the biggest challenge for lower secondary level teachers to include CT in their classrooms (Bocconi et al., 2022). We hope that the teacher trainings we offer in combination with our handbook and openly accessible videos created by the project will help to overcome this challenge in the future.

Teacher trainings are designed as in-person day-long workshops offered regularly by the project consortium. The training comprises four modules, best taken in sequence and within close proximity (e.g. weeks or months). The schedule of the teacher training is designed such that the teacher can familiarize themselves firstly with the technology from the students' perspective (figure 4). After having seen the *Task Families* and the AR-view in action they will start to learn about the web portal and how to design meaningful learning paths. In the following sessions the focus will be on embedding <colette/> in the classroom, be it the creation of learning paths or the usage of the soon to be published digital classroom.

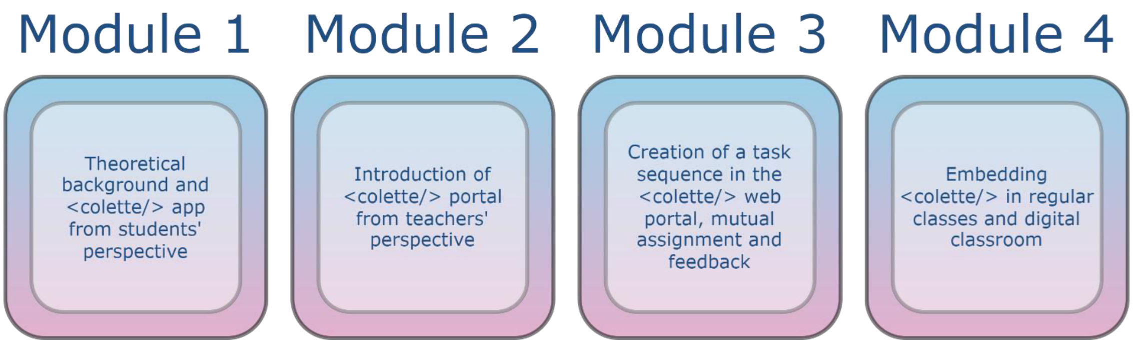

Module 1
Theoretical background and <colette/> app from students' perspective
Module 2
Introduction of <colette/> portal from teachers' perspective
Module 3
Creation of a task sequence in the <colette/> web portal, mutual assignment and feedback
Module 4
Embedding <colette/> in regular classes and digital classroom

Figure 4: The four modules of the teacher training. The educator starts with the students' perspective, gets to know the authoring tool (the web portal) and starts creating meaningful learning paths. Lastly, educators will learn about the possibilities on how to embed <colette/> in their regular classes.

HOW THE PILLARS WORK TOGETHER

After having seen what the pillars are meant for and which area of the learning environment they are defining, we are now looking at their interactions and how they are the base for <colette/> (figure 4).

The teacher can attend one of our teacher trainings to know better what CT is and how they can integrate this concept in their lessons using <colette/>. The educator has, hence, learned how to use the web portal and create meaningful CT learning paths.

The first and second pillar, so the app and the web portal, are the ones actively being used in and for the lessons. The educator uses the web portal to create learning paths which will then be worked at by the students using the app.

The third pillar, the didactic content, is mainly created by the project consortium and is then adapted by the educators to fit their curriculums best. Easy-to-use material is one of the teachers' requests for a better inclusion of CT in their classrooms (Borkulo et al., 2020). For this matter are the tasks presented as blueprints which makes it easy to use. The educators can adapt this material to their liking and curriculum.

OUTLOOK

Having described the four pillars, their interaction and concepts behind them, we hope that the communication of the design of this learning environment will be useful for conceptualizing similar digital learning environments, being it for CT tasks or for fostering other skills. In our point of view, the teacher training is not only a good dissemination activity but it is an important and valuable resource for feedback. Teachers and their students are the target group so we need to have their needs in mind implementing and designing the learning environment.

In the remaining project lifetime, we plan on adding more *Task Families* (like interpolation of functions or linear patterns) in the portal and as unplugged *Task Families*. This way we are opening <colette/> to a broader spectrum of school subjects. Besides more *Task Families*, more assignment types following the idea of *Generic Tasks* will be added as well, using the potential of connecting the idea of *Task Families* and *Generic Tasks*.

A feature called Digital Classroom will be included in March 2023 and will help teacher including <colette/> in their offline classrooms. In this digital classroom students can send teachers their solutions, asking questions and the teacher can see the progress students are making. In the next upcoming months several teacher trainings in the project countries will be conducted as well as a teacher conference in July in Linz.

More information about the project and its progress can be found on the website colette-project.eu.

ACKNOWLEDGMENTS

The project is (partially) funded by the Erasmus+ grant program of the European Union (2020-1-DE03-KA201-077363). Neither the European Commission, nor the project's national funding agency PAD are responsible for the content or liable for any losses or damage resulting from the utilization of these resources.

REFERENCES

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., & Punie, Y. (2016). Developing Computational Thinking in Compulsory Education: Implications for Policy and Practice. *European Commission. Joint Research Centre for Policy Report 68*. https://data.europa.eu/doi/10.2791/792158.

Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M., Jasutė, E., Malagoli, C., Masiulionytė-Dagienė, V., Stupurienė, G., Inamorato Dos Santos, A., Cachia, R., Giannoutsou, N., and Punie, Y.. Reviewing Computational Thinking in Compulsory Education. *Luxembourg: Publications Office*, 2022. Web.

Borkulo, S P van, Kallia, M., Drijvers, P., Barendsen, E., and Tolboom, J.. (2020) Computational Practices in Mathematics Education: Experts' Opinions. *The 14th International Congress on Mathematical Education Shanghai*, 12th –19th July, 2020, 2020, 6.

Fraillon, J., Ainley, J., Schulz, W., Friedman, T., und Duckworth, D.. Preparing for Life in a Digital World: IEA International Computer and Information Literacy Study 2018 International Report. Cham: *Springer International Publishing,* 2020. https://doi.org/10.1007/978-3-030-38781-5.

Lopez, M., Whalley, J., Robbins, P., und Lister, R.. Relationships between Reading, Tracing and Writing Skills in Introductory Programming. *In Proceedings of the Fourth International Workshop on Computing Education Research,* 101–12. Sydney Australia: ACM, 2008. https://doi.org/10.1145/1404520.1404531.

Lyon, J. A., and Magana, A. J. Computational Thinking in Higher Education: A Review of the Literature. *Computer Applications in Engineering Education 28, Nr. 5* (September 2020): 1174–89. https://doi.org/10.1002/cae.22295.

Medová, J., Milicic, G., and Ludwig, M. Graph Problems as a Means for Accessing the Abstraction Skills. *Twelfth Congress of the European Society for Research in Mathematics Education (CERME12)*, Feb 2022, Bozen-Bolzano, Italy. ?hal-03748478

Milicic, G., Wetzel S., and Ludwig, M. Generic Tasks for Algorithms. *Future Internet 12, Nr. 9* (3. September 2020): 152. https://doi.org/10.3390/fi12090152.

Rottenhofer, M., Sabitzer, B., and Rankin, T. Developing Computational Thinking Skills Through Modeling in Language Lessons. *Open Education Studies 3, Nr. 1* (1. Januar 2021): 17–25. https://doi.org/10.1515/edu-2020-0138.

Stäter, R., Läufer, T., Ludwig, M. (2023). Teaching Computational Thinking with <colette/>. *In Proceedings of ROSEDA Conference*.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., und Wilensky, U. Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology 25, Nr. 1* (Februar 2016): 127–47. https://doi.org/10.1007/s10956-015-9581-5.

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, 62(8), 22–25. https://doi.org/10.1145/3341221

Wing, J.M. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35. https://doi.org/10.1145/1118178.1118215