Cooperative Thinking, or: Computational Thinking meets Agile

Marcello Missiroli
University of Bologna, Italy

Daniel Russo
University of Bologna, Italy

Paolo Ciancarini
University of Bologna, Italy

***Abstract*—In this paper, we propose the Computational Thinking concept, which is obtained by enhancing by merging the values of Computational Thinking and Agile. We analyze four existing teaching models for training Cooperative Thinkers, supported by experimental data, and propose an educational path that can promote the early development of this complex skill.**

I. Introduction

Literacy is an individual skill, needed by any citizen that interacts with society. The individual scope of literacy has deeply influenced teaching methodologies and especially students' evaluations, concentrating the educational effort on the individuals. For instance, a consequence of this is that some effort is spent in schools on overcoming individual differences among students [18].

A new form of literacy is Computational Thinking (ComT), a concept that has enjoyed much popularity during the last decade, especially in the educational field. ComT — not to be confused with programming ability — is usually considered an individual skill, and trained as such. An individualistic approach does not reflect current teaming structures of both science and business, where problems and projects have become so complex that a single individual cannot handle them within a reasonable time frame. To handle the increasing complexity, we need people able to act and operate as a team, which is more that the sum of individual skills [7].

This is already happening in the industry. Several companies rely upon teams for innovation and digital transformation, especially to solve "wicked problems" whose solutions are not provided by individuals but by self-organizing teams [8].

In Software Engineering (SE) a teaming concept is epitomized by Agile methodologies. The Agile "philosophy" acknowledges that not all information and know-how might be available at the beginning of a project; therefore, reaching the goal requires several iterations, each closer to the solution. A key factor is self-organization, meaning that any team member contributes with her knowledge, ability and technical skills in order to work out a solution.

We argue that Agile principles and values should enhance the current efforts to establish Computational Thinking as a fundamental literacy ability; we call such a combination Cooperative Computational Thinking, or **Cooperative Thinking** (CooT) in short.

This new skill requires a serious reimagining of education in general and the teaching approach to Software Engineering in particular, as educators should not only promote coding skills and provide knowledge, but also foster collaboration skills and train teams of students to cooperate on problems which are too difficult for them to solve individually. This is something that in our experience the current schooling model is not really ready to handle, especially after primary school.

Generally speaking, mainstream educational models have difficulties in handling cooperation, and especially so the farther we move away from primary school. The quality of education is often tied to fundamental skill expertise; one of the most recognized indicator is the result of the international PISA test, that evaluates effective a country has been at deploying their prescribed math, science, and reading curriculum. This is particularly true for the specific field of Computer Science (CS), where programming is considered an individualistic and personal skill.

We lack a general approach to enable group skills in this context. Even if this idea may be widely shared by the community, we did not find any evidence of a comprehensive approach to it. This is probably due to the lack of explicit awareness of such concept as enabler of Digital Transformation processes: we may use it implicitly without recognizing it.

However, in the future, "pure" knowledge might become less important, even to the point of becoming a commodity, and soft skills could raise in importance. An educational system focusing on hard, technical skills could have difficulties in promoting the other. As Zhao [21] pointed out, there is an inverse correlation between PISA test scores and entrepreneurial capacity, a measured by the Global Entrepreneurship Monitor (GEM), the worlds largest entrepreneurship study. Specifically, the countries with the top PISA scores had an average GEM:PISA ratio of less than half of the mid- and low-scoring countries.

The traditional educational paradigm is not tailored to educate people able to handle complex issues or *wicked problems* [2]; PISA-like evaluations are meaningless to determine the educational system's efficiency, since the only offers an evaluation of the *individual*. So, the gap between student's formal educational background and real life wicked and complex problems task becomes larger as the level of predictability decreases and uncertainties increases [15].

To help educators find their way to promote Agile teaming to SE and CS students, we analyzed processes and interactions in four different teaching methodologies that, in most

2377-570X/17 $31.00 © 2017 IEEE
DOI 10.1109/CSEET.2017.37

way, mirror standard development models: solo programmer, pair programmers, self-organized teams, and directed teams. We report differences, practical and educational issues, their relative strengths with respect to developing Computational Thinking skills on one hand and how they impact Agile team-related skills, that form the base of Cooperative Thinking, on the other.

We recommend using a mix of teaching strategies in any case, though self-organized teams seems to be the correct way to enact and support CoopT, as the final step of an educational path.

The rest of this paper is organized as follows.

- Section II provides information on current research.
- Section III presents our detailed proposal on Cooperative Thinking.
- Section IV substantiates our proposal by comparing four teaching models within the CS context in a Cooperative Thinking perspective.
- Section V discusses the results and proposes a general direction for CS teachers and outline future research.

II. Related work

Kickstarted by Wing's popular paper [20], Computational Thinking has generated a lot of interest in the scientific community in general. However more and more scholars argue whether the ComT concept is too vague to have a real effect.

Denning [6] claims that ComT is too vaguely defined and, most important in an educational context, its *evaluation* is very difficult to have practical effects. This same idea can be found in the CS Teaching community. Hoskey [9], for example, tries to decompose the ComT idea itself, in order to have an operative definition.

Though Agile development is eventually going mainstream in the professional world, *teaching* the Agile methodology is still relatively uncommon education, especially at the K-12 level, though things are slowly changing.

In general, programming is considered an individual skill and taught as such. Not many researchers challenge this idea, an in Carter's [3], and especially in Meier's works. We note however that the approach is hardly systematic, and no general consensus exists on how to proceed along this line.

III. Definition of Cooperative Thinking

More that a decade has passed since Jeannette Wing's paper conceptualizing Computational Thinking [20], and its influence is still strong. Even today, governments are realizing its importance, and update school programs worldwide (like the US initiative *"21st century skills"*— a welcomed change away from old educational policies that equated computer literacy in schools to productivity tools.

However, current educational approaches concentrate on coding (as an example, consider the *Hour of Coding* initiative), but this it not the end to it. Computational Thinking is made of complex, tacit knowledge, that overcomes limited resources and requires deep learning, lots of deliberate practice, and expert guidance. Coding is one aspect, and not necessarily the most important one.

Tasks solved by software systems are becoming more complex by the day, and many of these in the real world could be classified as *wicked problems* [16].

There is no single "best solution" to many such problems, but Pareto-optimal ones which may change over time — as is the case in the field of Science and Business. In this situation, satisfying expectations and requirements becomes harder and harder as they are beyond the limit of solvability for any single programmer.

Computational Thinking has been considered as the individual skill to solve problems, but it does not offer the variety of points of view required to solve difficult or wicked problems. Moreover, Computational Thinking (and its related programming skills) has traditionally been considered an individual skill, and taught as such. Teamwork and soft skills are generally not factored in, and even shunned as "cheating" in some introductory programming courses.

In our view, the general approach to ComT needs to be updated, by joining it with a pre–existing concept: Agile development values and practices.

The Agile Manifesto proposed a radically different perspective on software development, based on values that clashed with the established culture of time, based on linear hierarchies, top-down decision making and, in general, accepting the current system without voicing dissent or criticism. The most significant change is the paramount importance assigned to **communication and social interaction**, superseding the internal organizational rigidity, documentation, contracts, roles, and more.

In time, this led to the formalization important concepts (such as changing requirements, self-organizing teams, personal responsibility, ...) and programming practices (pair programming, test-first development, continuous integration, ...). Agile has proven in several contexts its usefulness, and it is now an established development model and its adoption is steadily growing.

Including some Agile principles and learning-as-execute experiences in training for Computational Thinking could be beneficial. We name this expanded definition of ComT as **Cooperative Thinking** (CoopT), defined as follows:

"Cooperative Thinking is the ability to describe, recognize, decompose problems and computationally solve them in teams in a socially sustainable way ".

This definition joins the basic values of both ComT and the Agile Manifesto; all these values are central not only for developers but also for educating individuals. As a result, Agile and Computational Thinking can complement each other, one providing logical reasoning, the other social skills and solid practices. Both arguably share the idea of evolving, incremental solutions and reflective practices based on critical thinking.

IV. Promoting Cooperative Thinking

To recap, CoopT is a an expanded concept of CompT that includes the social dimension, by way of . Similiarly to ComT, CoopT is a general theoretical concept that should be both understood and introduced as a learning goal. However, until now in CS teaching, only individual performances are usually evaluated in schools and colleges alike. Not the teamwork. We need to reinvent our approach to CS education starting from K-12 but also reinforcing it at university level if we are to overcome our future challenges. We need *thinking people* able to *coordinate effort* among themselves.

Since CoopT is a complex skill, it requires a series of coordinated activities that reinforce its two basic components (CoopT and Agile) that together build a CoopT mindframe. We built a general teaching model that linked four Learning Styles (as outlined by Kolb) to four established teaching models for CS: Individual learning, Paired learning, Directed group learning, and Self-determined group learning. We evaluated their impact by means of a series of experiment performed during the last few years on Italian high school programmers. Their curriculum include 17 hours/week of CS-related classes, thus resembling that of CS-undergraduates.

Even though research questions were not always straightforward, some interesting points were raised in favor of introducing Agile elements in education as a means to empower Computational Thinking and steer it toward Cooperative Thinking. We will also confirm some intuitions considering literature and personal experience as teachers.

A. Individual learning

Individual learning (also called Direct Instruction or Transmissive Education) is the oldest form of teaching, practiced everywhere in practically every subject. It is a simple, efficient model, and we all have plentiful experience of it. Its sequential progression is ideal for stimulating Computational Thinking. Once a concept is mastered, it allows to tackle more complex ones. Assessment is very easy, often automatic.

This successful model does, however, have several limitations. One important limitation is the fairness of the assessment, since students falling behind at the beginning of the course rarely have the capacity to catch up, as the time allotted for each student is the same for every student and more information. This situation has helped to create the so called "Ability Myth": it states that each of us is born with a set of abilities that hardly change during our lifetime.

Another drawback is the absence of positive social interaction. Direct teacher/student communication is limited by time, and student/student interaction, more often than not, results in direct competition or in nonconstructive and illegal help (i.e. cheating).

In our experiments [13], we tried to simulate a working day in a software house. Each student was given a moderately difficult task using a new work methodology (either TFD or User stories) within a limited time-frame. Without much surprise, both performance and the perceived utility of the activity mirrored their current skill level.

To summarize, individual learning help foster Computational Thinking but it is not useful (or maybe detrimental) to develop social skills needed for Cooperative Thinking. According to the Kolb's learning inventory [12], this teaching model better suits *Assimiliative* learners, since they like organized and structured understanding and respect the knowledge of experts. They also prefer to work by themselves.

B. Paired learning

Paired learning (also called *Dyadic Cooperative Learning Strategy* [17]) is also a very old active teaching technique but far less popular that the previous one. The basic principle involves the teacher posing a question or delineate a problem, then the students discuss in pairs and find their own way toward the solution; pair are switched often, sometimes even during the activity.

In the specific CS field, we find a natural transposition of this model in Pair Programming, one of the key Agile programming practices.

This model [5] has positive effects on retention, understanding, recall and elaborate skills at the cognitive and level and especially on mood and social skills; in the CS field, it also introduces the idea of software being an iterative, evolutionary and social process. Assessment is a more difficult than in the previous case (automatic evaluation is still possible, though).

We tested firsthand this effect in our experiments. We proposed the same methodology and problems stated in Sect. IV-A, but in this case we paired students according to the same performance classes described in Sect. IV-A. We had six possible pair types, homogeneous skill level pairs and non-homogeneous ones.

According to our results, homogeneous pairs performed generally equal or worse than their solo counterparts, but non-homogeneous pairs had statistically better results. In the latter case a form of epistemic curiosity [11] appeared, possibly unconsciously, and was a key motivating factor for the pair; the resulting interaction helped both to solve the task at hand and to develop social skills. Computational Thinking was also stimulated, but a little less than with the previous model, since the "effort" was split and each single task was not really challenging, requiring expertise more than logical reasoning.

To summarize, paired learning has beneficial effects on social skills related to Agile development, and generally is useful in leveling skills upwards. Knowledge building will however be much slower than in the traditional approach. This teaching model better suits *Convergent* learner types, since they want to learn by understanding how things work in practice, like practical activities and seek to make things efficient by making small and careful changes.

C. Directed group learning

Group learning is one of the many facets of Cooperative Learning, which is becoming fairly common in modern, constructivist-influenced education. In CS it is often paired with Project-Based Learning, proposing a complex task taken

from real-life with authentic evaluation, comprehensive of all phases of development [10].

Directed group models enforce linear hierarchies, top-down decision making, accepting the assumptions, acquiring all information in order to prepare a detailed plan and then following it — values that have also forged the way traditional education was conceived and in most cases is still carried out. In school projects, teachers assume the role similar to that of a Senior project leader, assigning tasks and roles to students according to their skill, knowledge, and ability and applying a certain degree of control. Assessing a group project is considerably more complex that both previous model, since it involves not only the final product, but also the process used and the interaction among the student and their relative contributions.

In a different experiment [14], we decided to give students a very challenging task, almost impossible to solve. They had to build from scratch a complete dynamic website, a task we estimated in about 30 man-hours to complete when handled by professionals. We only gave them 6 hours. This forced teams to make hard decision as to what was the most suitable course of action in order to make the best use of the allotted time and resources.To comply to the Waterfall development model, we provided the students with plenty of information (36 pages), roles, tight schedules.

From an educational viewpoint, the goal was definitely outside a single student's zone of proxymal development, but was theoretically doable as a team effort. From a different viewpoint, it resembles a wicked problem, since students lack all the knowledge and skills to complete the task, and should acquire them along the way [19]. The great amount of information and in general the directive role of the teacher gives the opportunity to put the accent on whatever learning goal is deemed important.

Results show that under these conditions, groups tend to concentrate on functional requirements and process-related goals instead of pursuing the real goal: delivering a working product to the "client". The products, on average, had very few working features, but they were hidden under a pleasant user interface, very close to the one proposed by the "management". Roles were followed rather closely (barring a few cases of internal dissent), timing was impeccable and even documentation was acceptable.

To summarize, this teaching model promotes the use social skills, while leaving the steering wheel in the hand of the teacher. This power can be used to provide a meaningful learning path, though slower that Individual Learning and with a non-trivial evaluation method. It also does not seem to stimulate enough other interesting skills, such as decision making. It better suits *Divergent* learner types, since they and will start from detail to logically work up to the big picture. They like working with others but like things to remain calm.

D. Self-directed group learning

This model is a different version of Group Learning, radically different than the previous one in that students have a strong degree of autonomy.In this case, the teacher becomes more of a guide and a facilitator, and invests a large amount of trust on the learners. This means that the teacher must **become part of the team** in order to maintain a high level of communication — making traditional assessment very difficult. Grades should therefore come from reflections, group and/or personal and peer evaluation, and must include an evaluation of teacher work, as any other team member. In CS terms, most of what has been said on Project-Based Learning in the previous subsection holds. In this case, the granted freedom can be a powerful weapon in the hands of the group, but it might also backfire.

In another experiment [14], we kept the same general structure outlined in IV-C, but we chose a simplified Scrum approach. The teams were given much less information and limitations; they only included the sprint length and a list of user stories. Everything else was to be decided by the team.

Results show that Agile teams performed generally better than their Waterfall counterpart in the same class with respect to overall product completions and number of featured delivered. This is not surprising, since Agile privileges the functional dimension over the non-functional ones. It is interesting to note that many chose challenging but interesting tasks, possibly failing along the way.

In general Scrum teams spent their effort to reach even difficult goals, whereas Waterfall teams "played safe", working on what they most comfortable with. Our interpretation is that the self-directed group model greatly promotes the use of social skills and other qualities relevant to Cooperative Thinking.It better suits *Accomodative* learner types, since they display a strong preference for doing rather than thinking. They do not like routine and will take creative risks to see what happens.

V. Conclusions

In this paper, we proposed a novel idea, Cooperative Thinking, that expands Computational Thinking in order to embrace the social qualities typical of Agile development. The proposal is graphically presented in Fig. 1.

We also outlined how four teaching models can be linked to four different Learning Styles that collectively, according to our experience, can help our students to build and reinforce all the basic skills that form the basis of CoopT.

However, effectively introducing *introducing* all these strategies in practical education is not easy. We assume that most CS courses are strongly oriented toward individual learning, the goal being to introduce and grasp the basic elements of CS and, specifically, programming; a short to medium-length programming project of average difficulty is usually included.

As soon as possible, elements of Pair Learning should also be presented. Specifically, Pair Programming should be introduced first and actively enforced as one of the main characteristics of class exercises throughout the course. Other elements of Agile-style programming could be introduced (such as Test-First Development, Continuous integration, ...) along with the necessary software tools (git, for example).

A project that verifies that students have indeed grasped such element should be simple in terms of programming complexity but balanced by process requirements, in that elements of Agile programming must be used and its use verified.

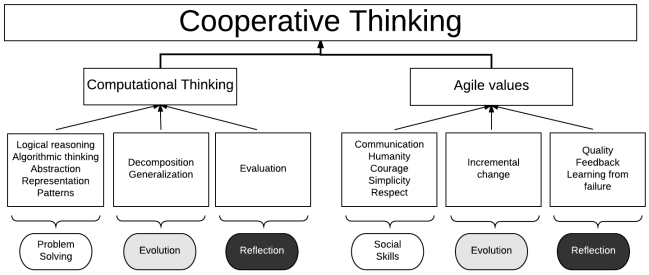

Fig. 1. Cooperative Thinking, Computational Thinking and Agile values breakdown (according to *Computing at School* [4] and Kent Beck [1])

Next, the group should become an important factor. We know that simply putting together people and telling them to work on a project is not enough to have an even decently efficient team. Preparation is in order, requiring some group-dynamic exercises, careful people selection, and some short project to test how the group works. Finally, a directed-team project of moderate to high difficulty and length should be realized by students.

The final step is, of course, that of proposing a demanding project to student teams and give them ample freedom. At this point student should have a solid programming knowledge and program development methodologies, a grasp of basic Agile practices, working experience with all necessary tools, and a team that knows its strength and weaknesses. This activity can actually be a course capstone project and should contribute significantly to the students' grade.

Our proposal requires formalization, testing and formal validation. Though every step is nothing new or complicated, the overall process is. Our research group is currently working on a comprehensive proposal and its field testing in both K-12 and university students for the next school and academic year.

It is our belief that Cooperative Thinkers will enjoy an edge on the job marketplace, making them more flexible, socially aware, and more able to handle future challenges, be they related to Computer Science or not.

ACKNOWLEDGMENTS

This work was partially funded under contract by the MIUR PRIN GAUSS project, the Institute of Cognitive Sciences and Technologies (ISTC) of the Italian National Research Council(CNR), and the Consorzio Interuniversitario Nazionale per l'Informatica (CINI).

REFERENCES

[1] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley, 2004.

[2] R. Buchanan. Wicked problems in design thinking. *Design issues*, 8(2):5–21, 1992.

[3] L. Carter. Ideas for adding soft skills education to service learning and capstone courses for computer science students. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 517–522, New York, NY, USA, 2011. ACM.

[4] A. Csizmadia, P. Curzon, M. Dorling, S. Humphreys, T. Ng, C. Selby, and J. Woollard. Computational thinking: A guide for teachers. 2015.

[5] D. F. Dansereau. Cooperative learning strategies. *Learning and study strategies: Issues in assessment, instruction, and evaluation*, pages 103–120, 1988.

[6] P. J. Denning. Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6):33–39, 2017.

[7] A. Edmonson. *Teaming to Innovate*. Wiley, 2013.

[8] A. Edmonson. Wicked Problem Solvers. *Harvard Business Review*, 94(June):52, 2016.

[9] A. Hoskey and S. Zhang. Computational thinking: what does it really mean for the k-16 computer science education community. *Journal of Computing Sciences in Colleges*, 32(3):129–135, 2017.

[10] W. Hung, D. H. Jonassen, R. Liu, et al. Problem-based learning. *Handbook of research on educational communications and technology*, 3:485–506, 2008.

[11] D. Johnson, R. Johnson, and K. Smith. *Active learning: Cooperation in the college classroom*. ERIC, 1998.

[12] D. A. Kolb. Learning styles inventory. *The Power of the 2 2 Matrix*, page 267, 2000.

[13] M. Missiroli, D. Russo, and P. Ciancarini. Learning agile software development in high school: an investigation. In *Proc. 38th ICSE*, pages 293–302. ACM, 2016.

[14] M. Missiroli, D. Russo, and P. Ciancarini. Agile for millennials: a comparative study. In *Proc. 1st Int. Workshop on Software Engineering Curricula for Millennials*, pages 47–53. IEEE Press, 2017.

[15] M. Raskino and G. Waller. *Digital to the Core: Remastering Leadership for Your Industry, Your Enterprise, and Yourself*. Routledge, 2016.

[16] H. Rittel and M. M. Webber. 2.3 planning problems are wicked. *Polity*, 4:155–169, 1973.

[17] R. Slavin. Cooperative learning. *Learning and Cognition in Education*, pages 160–166, 2011.

[18] K. Stanovich. Matthew effects in reading: Some consequences of individual differences in the acquisition of literacy. *Reading Research Quarterly*, pages 360–407, 1986.

[19] E. P. Weber and A. M. Khademian. Wicked problems, knowledge challenges, and collaborative capacity builders in network settings. *Public administration review*, 68(2):334–349, 2008.

[20] J. Wing. Computational Thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[21] Y. Zhao. *World class learners: Educating creative and entrepreneurial students*. Corwin Press, 2012.