Adopting Computer Science Pedagogical Patterns in Developing and Enhancing Computational Thinking Among Zero-Year Engineering Students

Loice Victorine ATIENO, TURCSÁNYI-SZABÓ Márta

Abstract Prospective undergraduate engineering students are assumed to possess Computational Thinking (CT) skills through basic programming. They are thus required to have considerable programming experience to enable them to execute tasks involving CT skills. Unfortunately, this is not the case as existing studies demonstrate lack of these skills, leading to poor academic performance. This was the case with first-year computer science international students at Eötvös Loránd University, faculty of Informatics, prompting the management to seek the causes of poor academic performance among the students and proper interventions. Two intervention courses were created to help develop and enhance CT among the students – *Introduction to Computational Thinking* and *Scratch Programming.* Apart from inculcating CT among learners, another underlying concern is that teachers are not keen to embrace CT approaches in the classroom. This is attributed to lack of time in utilising machine technology and lack of pedagogical skills. This was the case when the Scratch programming course was first offered. Like teaching any other subject, CT teachers require a collection of pedagogical experiences to enable effective CT teaching, including pedagogical patterns. Therefore, this paper demonstrates the adoption of computer science (CS) pedagogical patterns in the designing and teaching Scratch Programming course to zero-year students in the faculty to develop and enhance CT. The patterns were established through a literature review and tested in a course designed for zero-year computer science students for a period of 14 weeks. The patterns established through the literature review were deployed in the development and delivery of the course. The survey was conducted to establish the effectiveness of the tool. This formed part of preliminary results for the ongoing design research on integration of CT in creative learning.

Keywords: Computational Thinking, Pedagogical Patterns, Computer Science Pedagogical Patterns, Scratch Programming Course, Engineering Students

1. Introduction

Instituting computing and computational thinking (CT) in the school curriculum has triggered excitements and challenges related to new subjects [1]. This, therefore, necessitates the need for teachers to adopt new suitable subject delivery pedagogies especially, those related to algorithms, programming, and development of CT [2]. Despite substantial efforts to improve the understanding of CT, teaching CT for problem-solving is a new field for most teachers coupled with the complexity of problem-solving arising from executing new subject matter [3]. Teachers are also faced with challenges resulting from existing and emerging learning designs resulting from new ideas augmented by utilization of new technologies, with pedagogic theories and anecdotal explanations of other's practices being the only support available [4].

[1] Sue Sentance and Andrew Csizmadia, 'Computing in the Curriculum: Challenges and Strategies from a Teacher's Perspective', *Education and Information Technologies*, 22.2 (2017), 469–95 <https://doi.org/10.1007/s10639-016-9482-0>.

[2] Sentance and Csizmadia.

[3] Judith A. Cooper and Kristin L. Gunckel, *Teacher Perspectives of Teaching Computational Thinking* (Baltimore, Maryland, 2019).

[4] Diana Laurillard and Michael Derntl, 'Learner Centred Design - Overview', in *Practical Design Patterns for Teaching and Learning with Technology*, Yishay Mor (Rotterdam, Boston: Sense Publishers, 2014), pp. 13–16.

Implementing these theories and research can be challenging, hence the continuous research on practical methods of sharing successful and commendable practice amongst educators [5]. Numerous studies have been conducted on strategies and approaches for engaging learners in CT, focusing on creating new learning environments, tools, and activities that facilitate learning of CT [6]. Consequently, this has provoked the need for accessible, easy-to-use, and adaptable contextualised models and representations [7]. The representations and models (in form of case studies and learning design patterns) are effective tools and resources that facilitate educators' conceptualisation of innovative and alternative learning methods, particularly in complex tasks [8]. The critical question is, can CS pedagogical design patterns be employed in developing and enhancing CT?

Therefore, this paper seeks to demonstrate how CS pedagogical patterns were adopted in the development and implementation of a CT development course such as the Scratch programming Course.

2. Computational Thinking

CT supports capability advancement while lessening computing limitations as it involves thinking through problems and deploying the steps resulting to a solution [9]. CT is open to all and not just students of technology as it impacts learners' attitudes towards problem solving, ensuring success for the problem-solver. CT offers learners resources and aid for discovering new or distinctive problem-solving techniques hence boosting their confidence in problem solving. Teachers need to continually strive towards inculcating this sense of agency within their learners to shape their capacity to manage themselves in and out of the classroom and in the future. To facilitate the integration of CT, teachers' support is also essential. Teaching CT has since been perceived as mainly a constructionist idea [10].

According to constructionism, deep learning occurs when learners create their own purposeful projects together with other learners then meticulously reflect on the process while enabling freedom to explore their interests using technology [11]. Based on the constructionist approach,

---

[5] Robyn Philip, 'Finding Creative Processes in Learning Design Patterns', *Australasian Journal of Educational Technology*, 34.2 (2018), 78–94 <https://doi.org/10.14742/ajet.3787>.

[6] Mitchel Resnick and others, 'Scratch: Programming for All', *Communications of the ACM*, 52.11 (2009) <https://doi.org/10.1145/1592761.1592779>; Pratim Sengupta and others, 'Integrating Computational Thinking with K-12 Science Education Using Agent- Based Computation : A Theoretical Framework', *Educ Inf Technol*, 18 (2013), 351–80 <https://doi.org/10.1007/s10639-012-9240-x>.

[7] Sue Bennett and others, 'Learning Designs: Bridging the Gap between Theory and Practice', in *In ICT: Providing Choices for Learners and Learning. Proceedings Ascilite Singapore 2007.*, 2007, pp. 51–60; Matt Bower, 'Design Thinking and Learning Design', in *In Design of Technology-Enhanced Learning Integrating* (Bingley: Emerald Publishing., 2017), pp. 121–58; Peter Goodyear and Symeon Retalis, 'Learning, Technology and Design', in *Technology-Enhanced Learning: Design Patterns and Pattern Languages*, ed. by Peter Goodyear and Symeon Retalis, 2nd edn (Rotterdam, Boston: Sense Publishers, 2010), pp. 1–27.

[8] Philip.

[9] J. M. Wing, 'Computational Thinking', *Communications of the ACM*, 49(3).March 2006 (2006), 33–35 <https://doi.org/10.1109/vlhcc.2011.6070404>.

[10] Marina Umaschi Bers and others, 'Computational Thinking and Tinkering : Exploration of an Early Childhood Robotics Curriculum', *Computers & Education*, 72 (2014), 145–57 <https://doi.org/10.1016/j.compedu.2013.10.020>; Alan Buss and Ruben Gamboa, 'Teacher Transformations in Developing Computational Thinking : Gaming and Robotics Use in After-School Settings', in *Emerging Research, Practice, and Policy on Computational Thinking*, ed. by P. J. Rich and C. B. Hodges, 2017, pp. 189–203 <https://doi.org/10.1007/978-3-319-52691-1>.

[11] Bers and others.

various CT instructional approaches and techniques have been identified that can be used at different levels and across the curriculum [12]. This includes teachers using CT to model their own comprehension, learning, and progress (modelling) [13]; Collaboration among teachers to accomplish interdisciplinary projects (integration) [14]; teachers releasing responsibility gradually; teachers encouraging learners and giving tips and hints for problem-solving using probing questions rather than giving real solutions [15]; and using CT vocabulary across the curriculum [16]. Apart from the CT teaching approaches, teaching and learning in formal learning is structured. Various CT development models have been proposed that stipulate the CT to be developed and the process of CT development.

This is illustrated in "*A model for developing computational thinking skills*" [17] using a three-staged problem-solving cycle, dependent on CT to solve problems algorithmically: *problem definition* (problem formulation, abstraction, problem reformulation, and decomposition); *problem-solving* (data collection and analysis, algorithmic design, parallelization and iteration, and automation); and *analysing the solution* (generalization, testing, and evaluation) as shown in figure 1. The problem-solving process is a cycle starting from problem formulation and ending with evaluation of the whole process. CT is viewed as a problem-solving approach that requires creativity and critical thinking for the desired outcomes to be realised.

Despite the extensive research on the approaches, tools, and even activities for developing and enhancing CT in the classroom, an underlying concern is that teachers are not keen to embrace CT approaches in the classroom [18]. This is attributed to a lack of time in utilising machine technology, combined with the lack of pedagogical skills [19]. Like teaching any other subject, CT teachers require a collection of pedagogical experiences to enable effective CT teaching, including pedagogical patterns.

---

[12] Enoch Hunsaker, 'Computational Thinking', in *The K-12 Educational Technology Handbook*, ed. by A. Ottenbreit-Leftwich and R Kimmons (EdTech Books, 2020), pp. 1–16 <https://edtechbooks.org/k12handbook/computational_thinking>; Abeera P Rehmat, Hoda Ehsan, and Monica E Cardella, 'Instructional Strategies to Promote Computational Thinking for Young Learners', *Journal of Digital Learning in Teacher Education*, 36.1 (2020), 46–62 <https://doi.org/10.1080/21532974.2019.1693942>.

[13] Rehmat, Ehsan, and Cardella.

[14] Bers and others.

[15] Buss and Gamboa; Rehmat, Ehsan, and Cardella.

[16] Aman Yadav and others, 'Computational Thinking in Elementary and Secondary Teacher Education', *ACM Trans. Comput. Education*, 14.1 (2014), 16 <https://doi.org/http://dx.doi.org/10.1145/2576872 1.>; Hunsaker.

[17] Tauno Palts and Margus Pedaste, 'A Model for Developing Computational Thinking Skills', *Informatics in Education*, 19.1 (2020), 113–28 <https://doi.org/10.15388/INFEDU.2020.06>.

[18] Nur Hasheena Anuar, Fitri Suraya Mohamad, and Jacey-lynn Minoi, 'Contextualising Computational Thinking : A Case Study in Remote Rural Sarawak Borneo', *International Journal of LEarning, Teaching and Educational Research*, 19.8 (2020), 98–116.

[19] C. C. Selby, 'How Can the Teaching of Programming Be Used to Enhance Computational Thinking Skills? (Doctoral Dissertation, University of Southampton, England, United Kingdom). Https://Doi.Org/10.1016/j.Jsv.2010.04.020' (University of Southampton, England, United Kingdom), 2014) <https://doi.org/https://doi.org/10.1016/j.jsv.2010.04.020>.

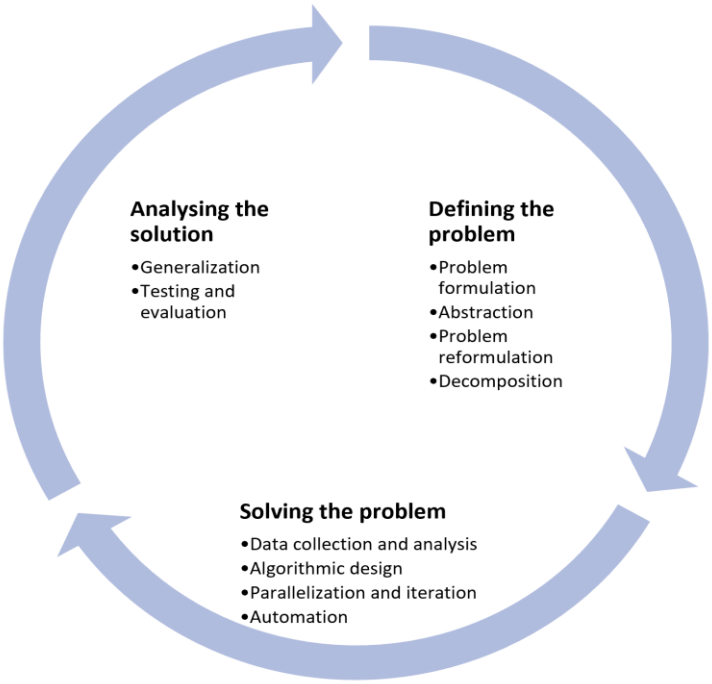

Figure 1: A model for developing computational thinking skills [20]

3. Pedagogical Patterns

Pedagogical Pattern (a term coined by the Pedagogical Patterns Project Team) endeavours to depict successful teaching practices to effectively share them with others as a means of learning from them [21]. Educational design has gained considerably from patterns especially in connection to: presenting the teacher-designer with an elaborate set of design initiatives; presenting the design ideas in a structured manner enabling understanding of relationships between design patterns; merging a clear expression of a design problem and solution, and recommending a justification which links pedagogical philosophy, research-based evidence and experiential knowledge of design; and presenting the knowledge in a way that it facilitates an iterative, fluid, process of design, extending over hours or days [22].

Pedagogical Patterns are not new ideas; rather, they are comprehensively tested and proven useful practices in various contexts by numerous people hence, they are not something *invented*, instead they are *discovered* [23]. The underlying assertion backing this effort is the tried and tested problem-

[20] Palts and Pedaste.

[21] Eva Magnusson, 'Pedagogical Patterns – a Method to Capture Best Practices in Teaching and Learning', in *Genombrottet Konferens 2006*, 2006 <https://www.lth.se/fileadmin/lth/genombrottet/konferens2006/PedPatterns.pdf>.

[22] Peter Goodyear, 'Educational Design and Networked Learning: Patterns, Pattern Languages and Design Practice', *Australasian Journal of Educational Technology*, 21.1 (2005), 82–101 <https://doi.org/10.14742/ajet.1344>.

[23] Magnusson.

solving approaches to solving repeated challenges or tackling ordinary requirements created by experienced educational practitioners [24].

A wide array of CS education pedagogical practices stem from teachers' expertise. Novice teachers find it hard and time-consuming to map these pedagogical practices to existing learning theories, hence having the teachers depending on their insight or pedagogies observed while students [25]. A pattern poses a problem and its solution together with the forces that must be employed enabling the solution suitable to the problem. The patterns seek to solve problems such as learner's motivation, choosing and sequencing teaching materials, evaluating students, among others. As the call for integration of CT into the curriculum grows wider, teachers need not only models, but also a collection of pedagogical patterns that can efficiently facilitate the development and enhancement of CT among learners. This paper looks at how CS pedagogical patterns can be utilised by teacher to develop and enhance CT in formal learning.

3.1 Computer Science Pedagogical Patterns

Patterns enable the design and delivery of a single course lasting one term or semester. Even though the emphasis is on CS, the patterns may be utilised in other fields as they aim to capture good practice in an organised manner, enabling their transfer to others, particularly to novice teachers [26]. The patterns include patterns for course preparation, course delivery, evaluation, feedback, among others.

Patterns Categories	**Brief Description**	**Sample Patterns**
Course Development	These patterns facilitate course preparation and choosing the material. They guide the comprehension of the course itself and not its delivery	New Pedagogy for New Paradigms, Need to Know, Abandon Systems, Early Bird, Spiral, Lazy Professor.
Course delivery (whole Course)	Guides the design on the organisation of the material and deciding on activities.	Early Bird, Spiral, Group Work, Lazy Professor, Active Student, Buddy System, Language Reinforces Paradigm, Write Over Read, General Concepts First, Study Groups, Reduce Risk, Stealth Instructor.
Course delivery (course or major topics introduction)	The idea of these patterns has to do with the first introduction of new material. How can you introduce new topics? What initial activities are appropriate?	Set the Stage, Lay of the Land, Visible Plan, Learn their Names, Fixer Upper, Occam, Read Before Write, Consistent Metaphor, High Leverage.

[24] Yishay Mor and Niall Winters, 'Design Approaches in Technology Enhanced Learning', *Interactive Learning Environments*, 15.1 (2007), 61–75 <https://doi.org/10.1080/10494820601044236>.

[25] Mary Lou Maher and others, 'Design Patterns for Active Learning', in *Faculty Experiences in Active Learning: A Collection of Strategies for Implementing Active Learning Across Disciplines* (University of North Carolina Press, 2020).

[26] Joseph Bergin, 'A Pattern Language for Course Development in Computer Science', 2002 <http://csis.pace.edu/~bergin/patterns/coursepatternlanguage.html>.

Course delivery (Evaluation & Feedback)	These patterns cover testing and student evaluation.	Fair Grading, Fair Project Grading, Fair Team Grading, Key Ideas Dominate Grading, Student Online Portfolios, Grade it Again Sam, Students Selected Activities, Trial Exam, Self-Test, Debrief After Activities, Peer Feedback. Communication: Rule of 1-Rule of 2, 24 by 7, Constant Feedback, Differentiated Feedback, Early Warning, Anonymous Mailbox, Ask Your Neighbour.
Course delivery (Dealing with problems)	Problems always occur in running any course. These patterns give some advice about being prepared for the inevitable.	Buffers, Prepare Equipment, Let the Plan Go, Debrief, Human Professor, Capture Everything.

Table 1: Sample Computer Science Pedagogical Patterns [27]

3.2 Pedagogical Patterns in Teaching Computational Thinking

Palts and Pedaste in "*A model for developing computational thinking skills*" [28], outline three stages of CT development, with each stage outlining the key tasks to be carried out: *problem definition* (problem formulation, abstraction, problem reformulation, and decomposition); *problem solving* (data collection and analysis, algorithmic design, parallelization and iteration and automation); and *analysing the solution* (generalization, testing and evaluation).

Problem definition: constitute all the CT skills that are needed before starting to solve a problem. Problem-solving begins with *(a) problem formulation,* which is realised by researching in order to understand the problem to be solved. This is followed by *(b) abstraction* as definition of key ideas through identifying and extracting relevant information is vital in formulating the problem. The key aspects realised during abstraction are then modelled. There is then need to *(c) reformulate the problem* into a familiar and solvable one. Lastly, the problem is *(d) decomposed* into manageable tasks.

Problem solving: This is the second stage based on the model and constitute all CT skills implicated in crafting the problem solution. Algorithmic problem solving requires (e) *data collection and analysis* followed by *(f) algorithmic design* which is a series of ordered steps then deployment of *(g) parallelization and iteration* leading to the process *(h) automation*.

Analysing the solution: This involves *(j) generalization*, which is applying the problem-solving process to a broader array of problems. The last task is *(j) evaluation and testing*, involving processes' outcomes analysis (assessment and acknowledgment) based on efficiency and resource utilization. This also entail systematic testing and debugging, efficiency and performance constraints, error detection, among others. In formal learning, the above tasks can therefore be accomplished is a structured manner, as depicted in the flowchart in Figure 2.

To enable the learners to successfully accomplish these tasks, teaching CT requires a selection of various teaching approaches [29]. There are moments teacher-centred approaches are useful in introducing concepts and model capabilities; nevertheless, learner-centred pedagogies are paramount for learners of computing to fuse their understanding, knowledge transfer, creativity

[27] Bergin.

[28] Palts and Pedaste.

[29] Mark Guzdial, 'Education: Paving the Way for Computational Thinking', *Communications of the ACM*, 2008, 25–27 <https://doi.org/10.1145/1378704.1378713>.

development, and facilitate peer learning [30]. Continued professional development should enhance teachers' CT pedagogies and adaptation approaches based on student needs [31]. Unfortunately, there is no consensus on strategies for teaching and assessing the level of CT development in learners [32]. Complete integration of CT into the existing curricula unquestionably poses significant challenges, particularly for teachers, as they may not be able to determine suitable pedagogies for teaching CT [33]. This is evident in the concern relayed by experts regarding the shortage of qualified teachers for the new curriculum delivery, especially when introducing new ideas and concepts [34]. Therefore, this calls for the need for experts' documented experience to inform of pedagogical patterns in enhancing teachers' capability in teaching CT.

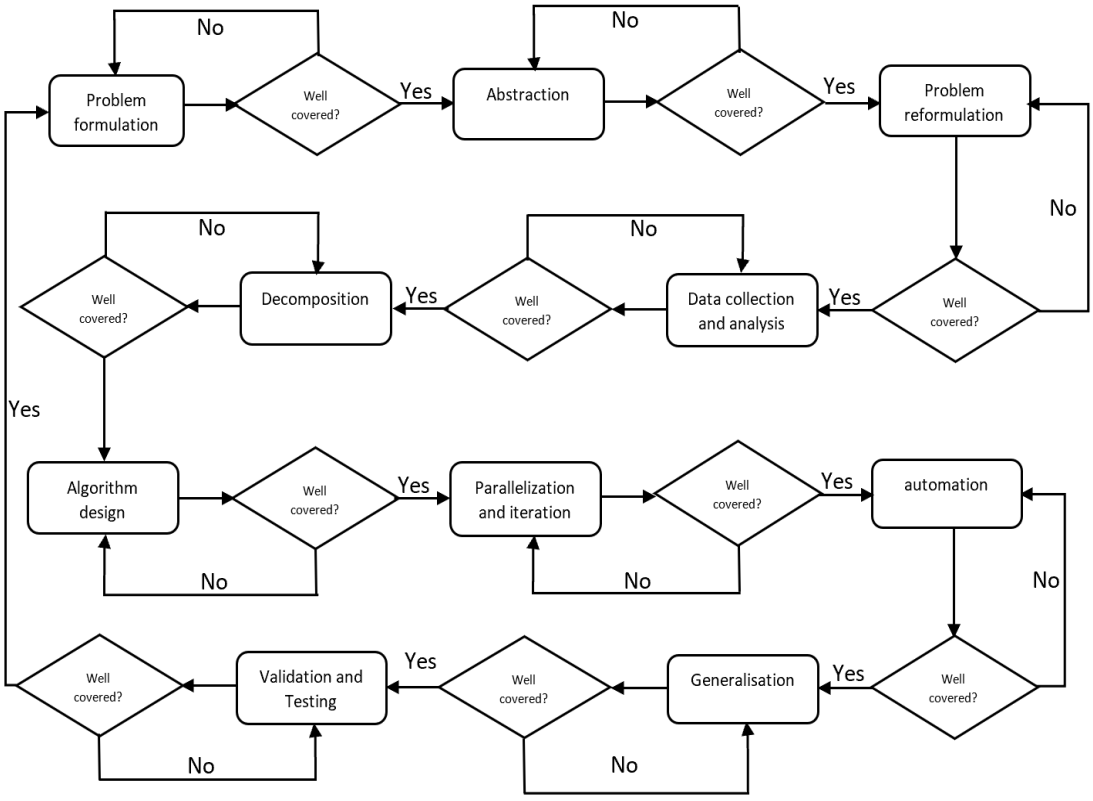

Figure 2: Flowchart Depicting the Process of Problem Solving with CT (Source: Author)

[30] Matt Bower, 'Redesigning a Web-Conferencing Environment to Scaffold Computing Students ' Creative Design Processes', *Journal of Educational Technology & Society*, 14.1 (2011), 27–42; Matt Bower and John G. Hedberg, 'A Quantitative Multimodal Discourse Analysis of Teaching and Learning in a Web-Conferencing Environment - The Efficacy of Student-Centred Learning Designs', *Computers and Education*, 54.2 (2010), 462–78 <https://doi.org/10.1016/j.compedu.2009.08.030>.

[31] Tara Stevens and others, 'Middle Level Mathematics Teachers' Self-Efficacy Growth through Professional Development: Differences Based on Mathematical Background', *Australian Journal of Teacher Education*, 38.4 (2013) <https://doi.org/10.14221/ajte.2013v38n4.3>.

[32] Karen Brennan and Mitchel Resnick, 'New Frameworks for Studying and Assessing the Development of Computational Thinking', *AERA*, 2012, 1–25 <http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf> [accessed 10 May 2019].

[33] Matt Bower and others, 'Improving the Computational Thinking Pedagogical Capabilities of School Teachers', *Australian Journal of Teacher Education*, 42.3 (2017), 53–72 <hhttp://files.eric.ed.gov/fulltext/EJ1137876.pdf>.

[34] Wen J. Peng and others, 'Emerging Perceptions of Teacher Quality and Teacher Development in China', *International Journal of Educational Development*, 34.1 (2014), 77–89 <https://doi.org/10.1016/j.ijedudev.2013.04.005>.

The patterns comprise of the context (where the pattern is applied), problem (problems to be solved), constraints and forces (what necessitate the application of the problem), and solutions (solution to the problem). The overall pattern here is the development and teaching of Scratch programming course

Context: Pedagogical patterns are applicable to courses or programs with complex tasks and ambitious goals and are beneficial to novice teachers [35]. In this case, Scratch programming course was offered to students with a vast background, had basic or no programming knowledge, and required CT skills to advance their studies in computer science engineering.

Problem: There is an assumption that students joining engineering courses at the undergraduate level possess CT skills through basic programming skills. Thus, they are expected to have considerable programming knowledge that would enable them execute projects that require CT skills. Unfortunately, existing studies conducted among undergraduate students exhibit lack of these skills [36], leading to poor academic performance. This has made entry-level courses complex for these students as they lack adequate programming knowledge (or background) as per the course prerequisites. The Scratch programming course students were international students from different countries with varied education systems; hence, their background, experience, and motivation are widely divergent. This made the harmonisation of student learning progress difficult, resulting to the increase in the complexity and difficulty of teaching the course. To attain the eventual goal of the course, there is need for a pedagogical method that is not only able to draw students' interest but also to keep this interest during the entire period of the course hence the adoption of the existing pedagogical patterns.

Forces: Teaching CT calls for the balance between teacher-centred and student-centred approaches to learning. When designing the course content and the activities for the course, the teacher has to have in mind how the two approaches can be balanced for effective learning to occur. The Scratch programming Course consisted of several activities to be carried out by learners. In case of complex activities scaffolding was used to achieve the desired goal.

Solutions: Developing CT in formal learning environment like a university setup is structured and this is best elaborated by "*A model for developing computational thinking skills*" [37]**.** CT development in such an environment is teacher guided; nevertheless, the utilization of learner-centred pedagogies are vital for learners to fuse their understanding, knowledge transfer, creativity development and facilitate peer learning [38]. To enhance the acquisition of CT in such an environment, various pedagogical patterns are adopted during CT development. The successful completion of each activity in CT development is evaluated, leading to the ultimate learning goal.

[35] Zhen Jiang, Eduardo B. Fernandez, and Liang Cheng, 'P2N : A Pedagogical Pattern for Teaching Computer Programming to Non-CS Majors', in *PLoP '11 Proceedings of the 18th Conference on Pattern Languages of Programs Retrieved From*, 2011.

[36] Gábor Csapó, 'Placing Event-Action-Based Visual Programming in the Process of Computer Science Education', *Acta Polytechnica Hungarica*, 16.2 (2019), 35–57; Mária Csernoch and others, 'Testing Algorithmic Skills in Traditional and Non-Traditional Programming Environments', *Informatics in Education*, 14.2 (2015), 175–97 <https://doi.org/10.15388/infedu.2015.11>.

[37] Palts and Pedaste.

[38] Bower, 'Redesigning a Web-Conferencing Environment to Scaffold Computing Students ' Creative Design Processes'; Bower and Hedberg.

In the Scratch Programming course, the development of CT was done using different learning themes. Various activities were carried out, which included lecturing, group discussion, teacher-led class discussions, project development, peer reviewing, reflections and evaluation. Our solution focused on the achievement of the overall goal which is to develop and enhance CT among learners.

3.3 Sample Implementation of the Patterns in Teaching Scratch Programming Course

Course Development: The course content was developed in themes, as shown in Table 2 adopted from creative computing with scratch curriculum [39], scaled down to meet our requirements. The learners were expected to come up with their own projects based on the themes provided.

Topic	Description	Required Sessions
Introduction to Scratch	Introducing creativity in computing and Scratch using sample projects and hands-on experiences.	4
Exploring Arts	Exploring arts by creating projects that include elements of music, drawing and dance.	4
Digital Stories Telling	Exploring storytelling by creating projects that include characters, scenes and narrative.	4
Developing Games	Exploring games by creating projects that define goals, levels and rules	6
Final project	Developing independent projects by first identifying the suitable project, utilising problem solving with CT, collaborating with others to improve the project and presenting the project and its development process.	8

Table 2: Course Themes

Various CS pedagogical patterns were used as a guide in developing the Scratch programming course with some of the sample patterns described below. According to Bergin [40], there is need for patterns that can facilitate course preparation and choosing of the materials to be taught. This is basically about the course and not its delivery. While developing the CT course, there was **Need to Know** the content and the emphasis given to each topic. This is important as some topics are more challenging, longer or even more interesting than others. The Scratch programming course was structured in a thematic manner that gave the learners a range of projects to work on. This also calls for structuring the content to enable proper content completion. Essential ideas in the course were mined and placed first in the course structure hence the **Early Bird. Spiral** was necessary during the course development to attain the necessary information on the important topics. This was a practical course with several projects to be accomplished with limited time available. Therefore, it was important to plan the activities aiming at what the learners were to do and not what the teacher would do, hence **Lazy Professor**. The **Abandoned Systems** approach was adopted that allowed materials selection and methodologies that enable learners' improvement on various fronts, resulting in solving new and meaningful problems. For novice teachers, teaching

[39] Karen Brennan, Michelle Chung, and Jeff Hawson, 'Creative Computing: A Design-Based Introduction to Computational Thinking', *Nature*, 2011, 73 <http://scratched.gse.harvard.edu/sites/default/files/curriculumguide-v20110923.pdf>.

[40] Bergin.

something new can be challenging hence need to adopt **New Pedagogy for New Paradigms** that advocates for use of different pedagogy, not just different examples, to teach the new concept.

Course delivery: Design-based learning approach was used to deliver the course content. The design-based learning approach is a method known for involving learners in solving real-life design problems as they reflect on the process of learning [41]. It is characterized with: the creation of artifacts, personalization of the creations, collaboration and reflection [42] which form the core of our learning process. For each theme, the students were introduced to the new concepts, and they were expected to create projects that would help develop and enhance CT among them.

Organising the course materials and activities: In most courses there is more to be taught and less time available and this was the case with the Scratch programming course that expected students to master various CT skills and dispositions within a period of 12 weeks. Teachers are therefore advised to use examples and exercises that cover more than one idea or topic simultaneously, hence **Multi Pronged Attack**. The course used design-based approach to learning where students were expected to develop projects. The approach was also supported by the **Active Student** pattern as each project developed covered various ideas and concepts that needed to be mastered. The development of projects kept the students active both in and out of class. The course was structured into themes and the learners' given opportunities to come up with their own projects within the themes (**Students Selected Activities**).
Introducing the course or new topic: **Set the Stage** when introducing new topics and at the beginning of the course to get the learners' attention and make sure all are ready to learn. This was done by naming the topic for emphasis and reinforcement; reviewing prerequisites; and showing the target using the learning objectives. A **Lay of the Land** provided a dramatic picture of the target. A **Visible Plan** about the course was created and shared with the students at the beginning of the course.

Evaluation and feedback: Students were provided with constant **Feedback** to enable the students establish their understanding of what was taught. **Positive Feedback First** was used as criticisms help the students enhance the criticized facets and should be given in time as it can significantly increase motivation [43]. To ensure students have understood the topic, they were given a chance to try what had been taught on their own (**Try it Yourself**). The students were also encouraged to be less dependent on the teacher through the use of **Student Portfolio** and **Peer Grading**. To ensure fair grading **Key Ideas Dominate Grading** was used and this was made possible using rubrics. Lastly, **Anonymous Feedback** was solicited using questionnaires on the effectiveness of the teaching style.

Dealing with problems: when students breeched the classroom code of conduct instead of punishing them, the vice was turned into a learning activity (**Human Professor**) for example if they missed lessons, they were given activities to accomplish that will cover the lesson taught in their absence.

[41] Mehalik and Schunn, 'What Constitutes Good Design? A Review of Empirical Studies of Design Processes', *International Journal of Engineering Education*, 22.3 (2006), 519–32.

[42] Brennan, Chung, and Hawson.

[43] Bergin.

Implication: By adopting CS pedagogical patterns in teaching Scratch programming course, the major conclusion of the work is that the existing pedagogical patterns can be used in teaching new courses. This was evident in how the learners improved in various areas, such as their motivation towards the course, persistence in dealing with complex tasks, managing individual and group learning, and embracing feedback.

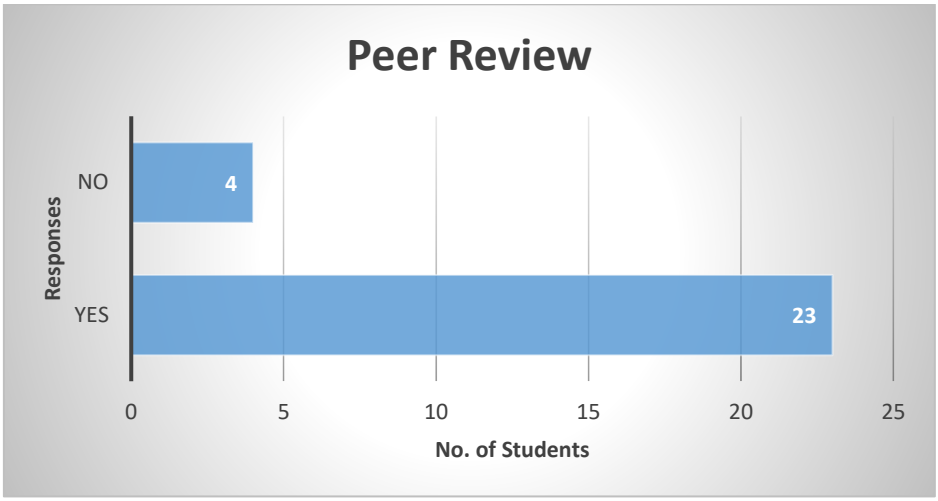

Figure3: Effectiveness of Peer Review

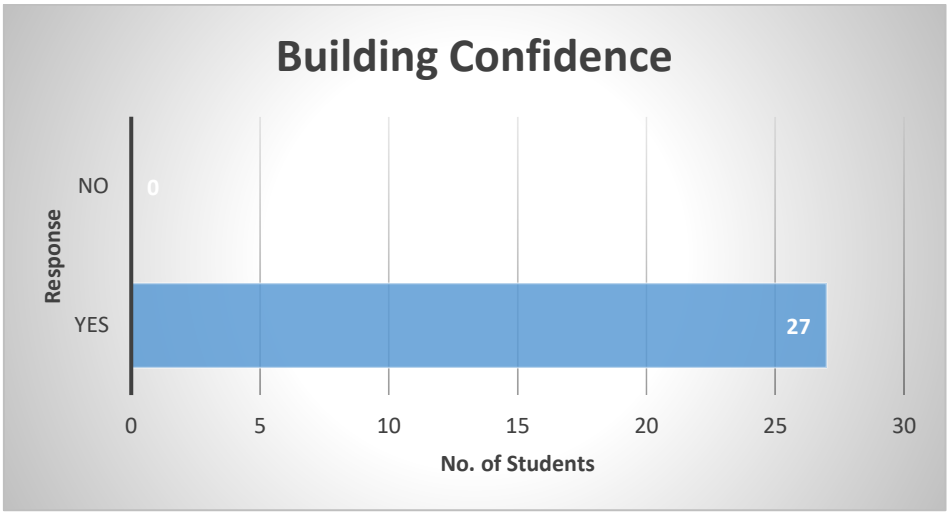

Figure 4: Building Student Confidence (Author)

Figures 3 and 4 shown the effectiveness of the design patterns on the learning outcome that has led to the development of CT. The patterns also helped in the selection of various learning activities, giving the learners a wide range of activities to select from (Figure 5), with most students preferring group activities. According to constructionism, deep learning occurs when learners create their own purposeful projects together with other learners then meticulously reflect on the process while enabling freedom to explore their own interests using technology [44]. The course was rated highly by the students (Figure 6). The sample results demonstrate the successful use of the patterns in the selected course and this can be applied in any other course.

[44] Bers and others.

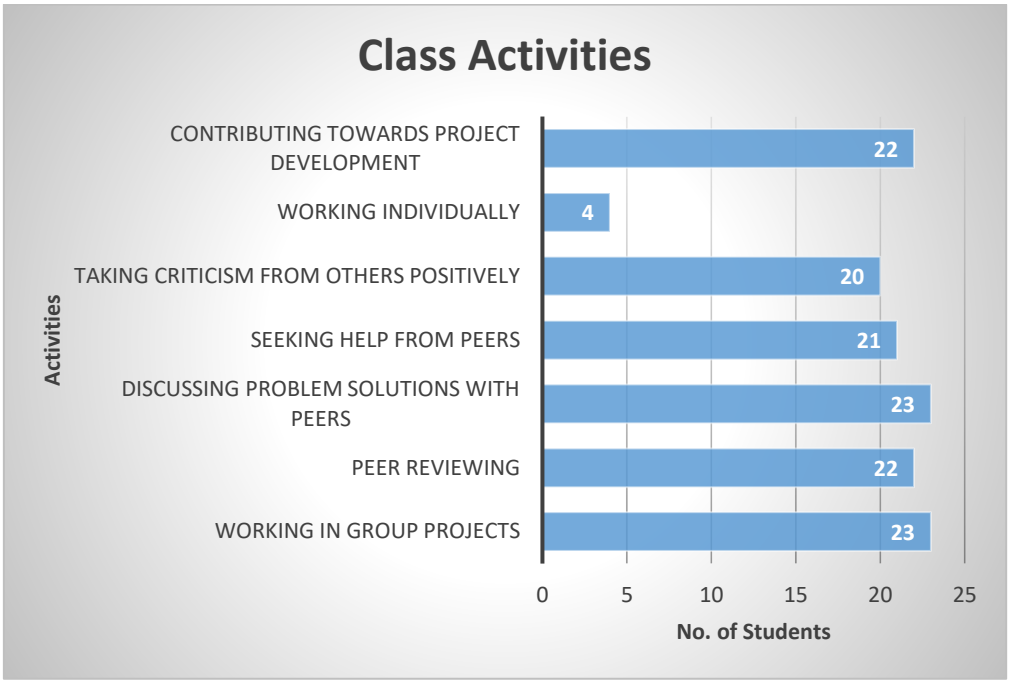

Figure 5: Learners' preference of activities (Author)

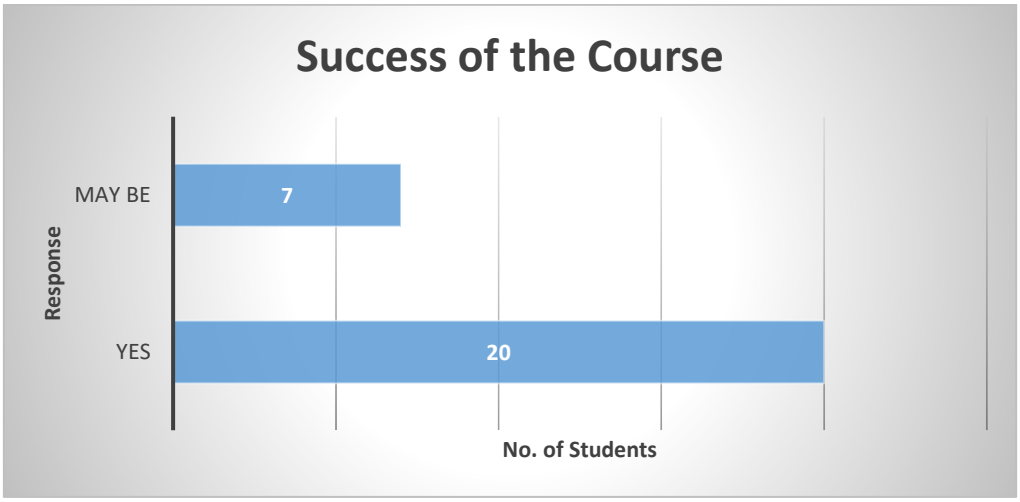

Figure 6: Overall effectiveness of the course (Author)

5. Conclusion and Recommendations

The goal of this paper was to share how various CS pedagogical patterns were used in the implementation of Scratching programming course to effectively develop and enhance CT among the learners in formal learning. Designing the course and eventually delivering it was guided by some of the sample pedagogical patterns outlined in section 2.3. Adopting the pedagogical patterns has led to the successful development of the course and its delivery. The research therefore advocates for the adoption of pedagogical patterns more so by novice teachers and enhancement of the patterns to enable development of CT in informal learning too. The patterns are also crucial in the in effective teaching of new courses should therefore be part of the teacher's tools for teaching.

References

1. N. H. Anuar, F. S. Mohamad, and J. Minoi: *Contextualising Computational Thinking : A Case Study in Remote Rural Sarawak Borneo*, International Journal of LEarning, Teaching and Educational Research, 19.8 (2020), 98–116. https://doi.org/10.26803/ijlter.19.8.6
2. S.Bennett, S. Agostinho, L. Lockyer, L. Kosta, J. Jones, R. Koper, and B. Harper: *Learning Designs: Bridging the Gap between Theory and Practice*, in In ICT: Providing Choices for Learners and Learning. Proceedings Ascilite Singapore (2007), pp. 51–60
3. J. Bergin: *A Pattern Language for Course Development in Computer Science*, (2002) http://csis.pace.edu/~bergin/patterns/coursepatternlanguage.html [Accessed on 15th March 2021]
4. M. U. Bers, , L. Flannery, E. R. Kazakoff, and A. Sullivan: *Computational Thinking and Tinkering : Exploration of an Early Childhood Robotics Curriculum*, Computers & Education, 72 (2014), 145–57 https://doi.org/10.1016/j.compedu.2013.10.020
5. M. Bower: *Design Thinking and Learning Design*, in In Design of Technology-Enhanced Learning Integrating, Bingley, Emerald Publishing, (2017), pp. 121–58 https://doi.org/10.1108/978-1-78714-182-720171008
6. M. Bower: *Redesigning a Web-Conferencing Environment to Scaffold Computing Students ' Creative Design Processes*, Journal of Educational Technology & Society, 14.1 (2011), 27–42
7. M. Bower and J. G. Hedberg: *A Quantitative Multimodal Discourse Analysis of Teaching and Learning in a Web-Conferencing Environment - The Efficacy of Student-Centred Learning Designs*, Computers and Education, 54.2 (2010), 462–78 https://doi.org/10.1016/j.compedu.2009.08.030
8. M. Bower, L. N. Wood, J. W. M. Lai, C. Howe, R. Lister, R. Mason, K. Highfield and J. Veal: *Improving the Computational Thinking Pedagogical Capabilities of School Teachers*, Australian Journal of Teacher Education, 42.3 (2017), 53–72 http://files.eric.ed.gov/fulltext/EJ1137876.pdf [Accessed on 2nd February 2021] https://doi.org/10.14221/ajte.2017v42n3.4
9. K. Brennan, , M. Chung, and J. Hawson, 'Creative Computing: *A Design-Based Introduction to Computational Thinking*, Nature, (2011) http://scratched.gse.harvard.edu/sites/default/files/curriculumguide-v20110923.pdf
10. K. Brennan, , and M. Resnick: *New Frameworks for Studying and Assessing the Development of Computational Thinking*, AERA, (2012), 1–25 http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf [Accessed on 19th March 2021]
11. A. Buss and R. Gamboa: *Teacher Transformations in Developing Computational Thinking : Gaming and Robotics Use in After-School Settings*, in Emerging Research, Practice, and Policy on Computational Thinking, ed. by P. J. Rich and C. B. Hodges, (2017), pp. 189–203 https://doi.org/10.1007/978-3-319-52691-1
12. J. A. Cooper and K. L. Gunckel: *Teacher Perspectives of Teaching Computational Thinking,* Baltimore, Maryland, (2019)
13. G. Csapó: *Placing Event-Action-Based Visual Programming in the Process of Computer Science Education*, Acta Polytechnica Hungarica, 16.2 (2019), 35–57 https://doi.org/10.12700/APH.16.2.2019.2.3
14. M. Csernoch, , P. Biró, J. Máth, and K. Abari: *Testing Algorithmic Skills in Traditional and Non-Traditional Programming Environments*, Informatics in Education, 14.2 (2015), 175–97, https://doi.org/10.15388/infedu.2015.11
15. P. Goodyear: *Educational Design and Networked Learning: Patterns, Pattern Languages and Design Practice*, Australasian Journal of Educational Technology, 21.1 (2005), 82–101,

https://doi.org/10.14742/ajet.1344
16. P. Goodyear and S. Retalis: *Learning, Technology and Design*, in Technology-Enhanced Learning: Design Patterns and Pattern Languages, ed. by Peter Goodyear and Symeon Retalis, 2nd edn, Rotterdam, Boston, Sense Publishers, (2010), pp. 1–27
17. M. Guzdial: *Education: Paving the Way for Computational Thinking*, Communications of the ACM, (2008), 25–27, https://doi.org/10.1145/1378704.1378713
18. E. Hunsaker: *Computational Thinking*, in The K-12 Educational Technology Handbook, ed. by A. Ottenbreit-Leftwich and R Kimmons, EdTech Books, (2020), pp. 1–16, https://edtechbooks.org/k12handbook/computational_thinking [Accessed on 20th July 2021]
19. Z. Jiang, E. B. Fernandez, and L. Cheng: *P2N : A Pedagogical Pattern for Teaching Computer Programming to Non-CS Majors*, in PLoP '11 Proceedings of the 18th Conference on Pattern Languages of Programs, *(*2011) https://doi.org/10.1145/2578903.2579163
20. D. Laurillard and M. Derntl: *Learner Centred Design - Overview*, in Practical Design Patterns for Teaching and Learning with Technology, Yishay Mor, Rotterdam, Boston, Sense Publishers, (2014), pp. 13–16 https://doi.org/10.1007/978-94-6209-530-4_2
21. E. Magnusson: *Pedagogical Patterns – a Method to Capture Best Practices in Teaching and Learning*, in Genombrottet Konferens, (2006) https://www.lth.se/fileadmin/lth/genombrottet/konferens2006/PedPatterns.pdf [Accessed on 18th July 2021]
22. M. L. Maher, , N. Dehbozorgi, M. Dorodchi, S. Macneil, and S. Diego: *Design Patterns for Active Learning*, in Faculty Experiences in Active Learning: A Collection of Strategies for Implementing Active Learning Across Disciplines, University of North Carolina Press, (2020)
23. Mehalik, and Schunn: *What Constitutes Good Design? A Review of Empirical Studies of Design Processes*, International Journal of Engineering Education, 22.3 (2006), 519–32
24. Y. Mor and N. Winters: *Design Approaches in Technology Enhanced Learning*, Interactive Learning Environments, 15.1 (2007), 61–75 https://doi.org/10.1080/10494820601044236
25. T. Palts, , and M. Pedaste: *A Model for Developing Computational Thinking Skills*, Informatics in Education, 19.1 (2020), 113–28 https://doi.org/10.15388/INFEDU.2020.06
26. W. J. Peng, E. McNess, S. Thomas, X. R. Wu, C. Zhang, J. Z. Li, and H. S. Tian: *Emerging Perceptions of Teacher Quality and Teacher Development in China*, International Journal of Educational Development, 34.1 (2014), 77–89 https://doi.org/10.1016/j.ijedudev.2013.04.005
27. R. Philip: *Finding Creative Processes in Learning Design Patterns*, Australasian Journal of Educational Technology, 34.2 (2018), 78–94 https://doi.org/10.14742/ajet.3787
28. A. P. Rehmat, E. Hoda and M. E. Cardella: *Instructional Strategies to Promote Computational Thinking for Young Learners*, Journal of Digital Learning in Teacher Education, 36.1 (2020), 46–62 https://doi.org/10.1080/21532974.2019.1693942
29. M. Resnick, , J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, E. Rosenbaum, J. Silver, B. Silverman and Y. Kafai: *Scratch: Programming for All*, Communications of the ACM, 52.11 (2009) https://doi.org/10.1145/1592761.1592779
30. C. C. Selby: *How Can the Teaching of Programming Be Used to Enhance Computational Thinking Skills?*, Doctoral Dissertation, University of Southampton, England, United Kingdom, (2014) https://www.researchgate.net/publication/299464834_How_can_the_teaching_of_programming_be_used_to_enhance_computational_thinking_skills [Accessed on 8th August 2021]

31. P. Sengupta, , J. S. Kinnebrew, S. Basu, G. Biswas, and D. Clark: *Integrating Computational Thinking with K-12 Science Education Using Agent- Based Computation : A Theoretical Framework*, Educ Inf Technol, 18 (2013), 351–80 https://doi.org/10.1007/s10639-012-9240-x
32. S. Sentance, and A. Csizmadia: *Computing in the Curriculum: Challenges and Strategies from a Teacher's Perspective*, Education and Information Technologies, 22.2 (2017), 469–95, https://doi.org/10.1007/s10639-016-9482-0
33. T. Stevens, Z. Aguirre-Munoz, G. Harris, R. Higgins, and X. Liu: *Middle Level Mathematics Teachers' Self-Efficacy Growth through Professional Development: Differences Based on Mathematical Background*, Australian Journal of Teacher Education, 38.4 (2013) https://doi.org/10.14221/ajte.2013v38n4.3
34. J. M. Wing: *Computational Thinking*, Communications of the ACM, 49(3), (2006), 33–35 https://doi.org/10.1109/vlhcc.2011.6070404
35. A. Yadav, C. Mayfield, N. Zhou, S. Hambrusch, and T. J. Korb: *Computational Thinking in Elementary and Secondary Teacher Education*, ACM Trans. Comput. Education, 14.1 (2014), 16 https://dl.acm.org/doi/10.1145/2576872. https://doi.org/10.1145/2576872

Authors

Loice Victorine ATIENO
Eötvös Loránd University
e-mail: atienomunira04@gmail.com

TURCSÁNYI-SZABÓ Márta
Eötvös Loránd University
e-mail: tszmarta@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 4, Number 1. 2022.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.4.1.2963

License

Copyright © Loice Victorine ATIENO, TURCSÁNYI-SZABÓ Márta

Licensee CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE, Hungary. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license.

http://creativecommons.org/licenses/by/4.0/