Co-funded by the Erasmus+ Programme of the European Union

Module 8

Educational environments for CT: design and aspects of integration

Author: University of Turku (Finland)
Peter Larsson
Ashok Veerasamy

Reviewers:
Panagiotis Kosmas (CARDET),
Mart Laanpere (TLN)
Vaida Masiulionytė-Dagienė (VU)

External reviewers:
Andreas Mühling (Germany),
Filiz Kalelioğlu (Turkey)

Piloting:
CARDET (Cyprus), University of Turku (Finland), Vilnius University (Lithuania)

Design (icons):
Vaidotas Kinčius (Lithuania)

Module *outline* is based on the work within the project "Future Teachers Education: Computational Thinking and STEAM" (TeaEdu4CT). Coordination: Prof. Valentina Dagienė, Vilnius University, Lithuania. Partners: Vienna University of Technology (Austria), CARDET (Cyprus), Tallinn University (Estonia), University of Turku (Finland), Paderborn University (Germany), CESIE (Italy), Radboud University (Netherlands), KTH Royal Institute of Technology (Sweden), Ankara University (Turkey). The project has received co-funding by the Erasmus+ Programme KA2.

Contents

General overview and aim

This module is about how to create educational environments that support the integration of Computational Thinking (CT) with Science, Technology, Engineering, Arts and Mathematics (STEAM). The educational environments provide technology that scaffold the application of CT. STEAM education merges the liberal arts with STEM (Science, Technology, Engineering and Mathematics) to make the ensemble of subjects approachable for a wider student population and to foster creativity. While arts provide context and creativity for STEM application, CT affects how the subjects are practiced. CT is seen here as a general attitude and skill set that supports the integration of methods from computer science to the disciplinary practices.

The three perspectives (mathematics, engineering, and science) on Computer Science (CS) combined with a model of CT are used as a framework in this module. The framework aids the teacher in designing the use of educational technology to support the integration of CT and STEAM. Each perspective provides a way to integrate CT with corresponding subject. Technology (T in STEM) subject is here interpreted to be covered by the engineering perspective. The students learn to plan educational technology support using the three perspectives. In the course project students form teams to create a design for a learning intervention where CT integrated with STEAM is supported by educational technology. The role of arts (A in STEAM) in the design of the learning intervention is to give context, create engagement and allow room for creativity.

This module is intended for future teachers of STEAM subjects.

Target group and prerequisites

The target group consists of prospective teachers in STEM, STEAM or one of the STEAM subjects.

There are no prerequisites except for good knowledge of own primary subject(s) and pedagogical studies.

However, basic knowledge of programming is an asset.

Keywords

instructional design, educational technology, computational thinking, STEM, STEAM

Related competence frameworks

The contents of this module promote the following competences of the DigCompEdu competence model (**competences not covered are excluded**; see Module 1 Unit 3 (Valentina Dagiene) for complete description of the model):

1. Professional engagement	
1.3. Reflective practice	To individually reflect on, critically assess and actively develop one's own digital pedagogical practice and that of one's educational community.
	This module provides an explicit framework to think about using digital educational technology in the context of teaching CT and STEAM.
2. Digital Resources	
2.1. Selecting digital resources	To identify, assess and select digital resources for teaching and learning. To consider the specific learning objective, context, pedagogical approach, and learner group, when selecting digital resources and planning their use.
	There are many educational technologies and therefore in this module the emphasis is to learn to recognize patterns that indicate possibilities to utilize a particular technology.
2.2.Creating and modifying digital resources	To modify and build on existing openly-licensed resources and other resources where this is permitted. To create or co-create new digital educational resources. To consider the specific learning objective, context, pedagogical approach, and learner group, when designing digital resources and planning their use.
	Even though the basic educational technologies already exist there is usually a need to modify them to fit the intended purposes. In this module students learn first to plan the application of the technology and then how to implement it.
3. Teaching and Learning	
3.1. Teaching	To plan for and implement digital devices and resources into the teaching process, so as to enhance the effectiveness of teaching interventions. To appropriately manage and orchestrate digital teaching interventions. To experiment with and develop new formats and pedagogical methods for instruction.
	To succeed the application of educational technology must be embedded in the context of the topic and support the activities leading to learning. The practical part of this module is about integrating the educational technology with teaching and learning.

3.4. Self-regulated learning	To use digital technologies to support self-regulated learning processes, i.e. to enable learners to plan, monitor and reflect on their own learning, provide evidence of progress, share insights and come up with creative solutions.
	This module presents a theory which the prospective teacher can use to create innovative instructional designs that support self-regulated learning. The educational technology used should scaffold the learning but not provide readymade solutions that stifle creativity and take away the joy of discovery.
4. Assessment	
4.1. Assessment strategies	To use digital technologies for formative and summative assessment. To enhance the diversity and suitability of assessment formats and approaches.
	The educational technologies for CT and STEAM are aimed to support arriving at solutions either by creating or discovering. The results are created step at a time and therefore formative assessment can be applied. The result is usually more than the sum of its parts and therefore it is also fit for summative assessment.
5. Empowering Learners	
5.3.Actively engaging learners	To use digital technologies to foster learners' active and creative engagement with a subject matter. To use digital technologies within pedagogic strategies that foster learners' transversal skills, open learning to new, real-world contexts, involve learners themselves in hands-on activities, scientific investigation and complex problem solving, or in other ways that increase learners' active engagement and creative expression.
	This point could be a description of why liberal arts was added to STEM subjects. The CT perspective of the different subjects supports the application of digital technologies.
6. Facilitating Learners' Digital Competence	
6.5.Digital problem solving	To incorporate learning and assessment activities which require learners to identify and solve technical problems or to transfer technological knowledge creatively to new situations.
	Computational thinking is at its core a problem-solving method. It opens the possibility to apply information technology to form a solution. By creating interesting and engaging problems the teachers transfer their knowledge to their students.

Learning Outcomes (LOs) and Assessment Methods

A successful learner will…	Assessment Methods
recognize opportunities for applying CT in STEAM topics (analysis)	Peer review of a plan (based on PRADA) describing how CT could be applied to STEAM topic(s)
be able to choose appropriate educational technology supporting CT in STEAM (application)	Peer review of a plan (based on CT Perspectives for STEAM framework) of how to choose right technology supporting learning of STEAM topic(s)
be capable of designing a CT and STEAM based learning intervention using educational technology (creative)	Lecturers' assessment of instructional design project report and presentation

Module plan and didactical approaches

This module is an elaboration of the Code, Connect and Create (3C) professional development model (Jocius et al., 2020) which is aimed to support teachers in integration of disciplinary content and CT principles. Coding is here generalized to use of educational technology supporting CT and the disciplinary is interpreted in context of STEAM. 3C uses the PRADA model of CT (Dong et al., 2019) whose principles are defined in terms suitable for teachers which are not well versed in computing. While familiar vocabulary is good it might leave out knowledge that is part of the CT contextual background. In this module we sharpen the PRADA model concepts by introducing mathematics, engineering, and science perspectives of CS to aid in its application.

Unit outline

Unit 1 - Three computer science perspectives

- Computer Science (CS) is a combination of mathematics, engineering and science perspectives. The perspectives support the application of CS in these areas.

Unit 2 - CT Perspectives for STEAM framework

- Pattern Recognition, Abstraction, Decomposition and Algorithm design (PRADA) is used as a model for Computational Thinking (CT). The CS perspectives aid in recognition of patterns in STEAM topics where CT could be applied.

Unit 3 - Choosing educational technologies based on CT Perspectives

- The benefit of CT is the possibility to apply methods and tools of CS. With CT one can solve problems, automate tasks and understand phenomena using computational technology. Educational technologies combine computational tools with scaffolding that make the tools suitable for learning.

Unit 4 -Project 1/2: ID for CT/ET+ STEAM

- In this unit the prospective teacher learn to collaborate in creating instructional content using a conceptual model of STEAM. The CT perspectives framework allows them to spot opportunities where CT could be applied.

Unit 5 -Project 2/2: ID for CT/ET+ STEAM

- In this unit the prospective teachers continue the collaboration and design a learning environment supporting the instructional content. Planning the integration of educational technology supporting CT is central part of creating the learning environment.

Unit 6 - Project presentations

- Presentations of the intructional designs which combine CT, STEAM and educational technology

Unit activities

ECTS 1 = 27 hours (in Finland)

Unit 1 - Three computer science perspectives
Lecture
Introducing the three perspectives of computer science: 90 min

Activity 1.1 Homework
Looking at own topic from one of the computer science perspectives: 60 minutes

Unit 2 – CT Perspectives for STEAM framework

Lecture
CT Perspectives for STEAM framework: 90 minutes

Activity 2.1 Homework
Description of own topic using CT Perspectives for STEAM: 60 minutes

Activity 2.2 Homework
Three peer reviews: 90 minutes

Unit 3 – Choosing educational technologies based on CT Perspectives

Lecture
Educational technology for CT perspectives: 90 min

Activity 3.1 Homework
Planning educational technology support for own topic: 2 hours

Activity 3.2 Homework
Three peer reviews: 90 minutes

Unit 4 – Creating instructional content for CT integrated STEAM

Lecture
STEAM Teaching Model 1/2: 90 min

Activity 4.1
Course project part 1/2: 5 hours

Activity 4.2
Team discusses the plan with the supervisor: 30 min

Unit 5 – Designing the learning environment for CT integrated STEAM

Lecture
STEAM Teaching Model 2/2: 90 min

Activity 5.1
Course project 2/2: 5 hours

Unit 6 – Project presentations

Activity 6.1
Team presentations 2 hours

Units and activities

Unit 1 - Three computer science perspectives

One challenge with CT is that it fits STEM topics very well. It seems to be part of them, but if it is naturally already there where is the contribution (Pears, 2019). CT could be said to be the ability to spot opportunities for methods and tools from Computer Science (CS) combined with the knowledge of how they could be applied. For this to be possible a computational thinker would also need familiarity with CS which is in line with Wing's (2006) thoughts. However, there is no consensus of what CS is, instead it can be said to consist of three major perspectives: math, engineering, and science (Tedre, 2018). This seems to be fortunate because it already incorporates the views of STEM subjects (technology (T in STEM) subject is here interpreted to be part of the engineering view). In this unit CS is introduced through the lenses of the three perspectives.

CS is a young science. The first department was established 1962 in Purdue university. With the help of CS, computers and associated technology have developed fast from the first computer ENIAC in 1945, 1800 square feet (167 square meters) of floor space, to billions of interconnected mobile phones 2020, which fit in their owners' pockets. Before 1990 the research in the field could be divided into Computer engineering which focused on hardware, Computer science (meaning the theoretical part of CS) which investigated programming and Information systems which was about the business use of computers. After 1990 two more subfields emerged. Software engineering investigated the production of software. Enterprise information technology focused on managing and supporting organizational use of information technology. The number of research areas in CS is growing, from 12 in 1996 to 26 in 2005. Even though the field seems to be diverging there are also something common to all. CS is a unique combination of mathematics, engineering, and science.

Mathematics

The word computer in CS referred originally to a profession where the task was to perform calculations according to a given plan. Usually, the result of a single computer was combined with others to form the intended outcome. Even though mathematical proficiency was required from the human computers to calculate correctly the work was otherwise mostly mechanical. The computation had been split into clearly specified calculations which were repetitive in their nature. This was quite different from the work of mathematician who investigates properties of

mathematical systems or their applications. Mathematicians planned how the computers should do their work. The first computer machine was created to provide additional computing capacity. Some of the human computers continued as programmers of the machine.

Mathematics is central to CS and it can be said to form the theory of computing. The different fields of mathematics provide the concepts and methods needed in computations. A mathematical model of a general-purpose computing machine was invented before its physical counterpart. Different kinds of mathematical aids and computing machines have a long history and their own technical evolution. It was only by the invention of the modern computer (machine) that it can be said that the theoretical and practical merged. The theoretical model of a computer was made for mathematical purposes to find out if it could be deduced if a formula would deliver a result or not. The question was raised by a giant in mathematics David Hilbert and answered by a 24-year-old British student Alan Turing.

The question raised by Hilbert called for a solution to the Entscheidungsproblem (German, "decision problem"). The challenge was to device a general method that allowed to calculate if a formula in first order logic vas valid, in other words that it was provable. For this to be possible an explicit definition of the steps of the calculation, an algorithm, was needed. Turing set out to solve the problem by making a computational model of calculating with pen and squared paper. Instead of multiple lines on a paper the calculation could be made on a single infinite line. It was enough to use the numbers zero and one since any value could be encoded using several of those numbers. Only one digit per square was allowed and the calculation was made one square at the time. A human can rely on his mental faculties to calculate, but here those were substituted with explicit rules. The result was a description of a machine with an infinite squared paper tape, a reading/writing head and rules that controlled its execution (see Figure 1).

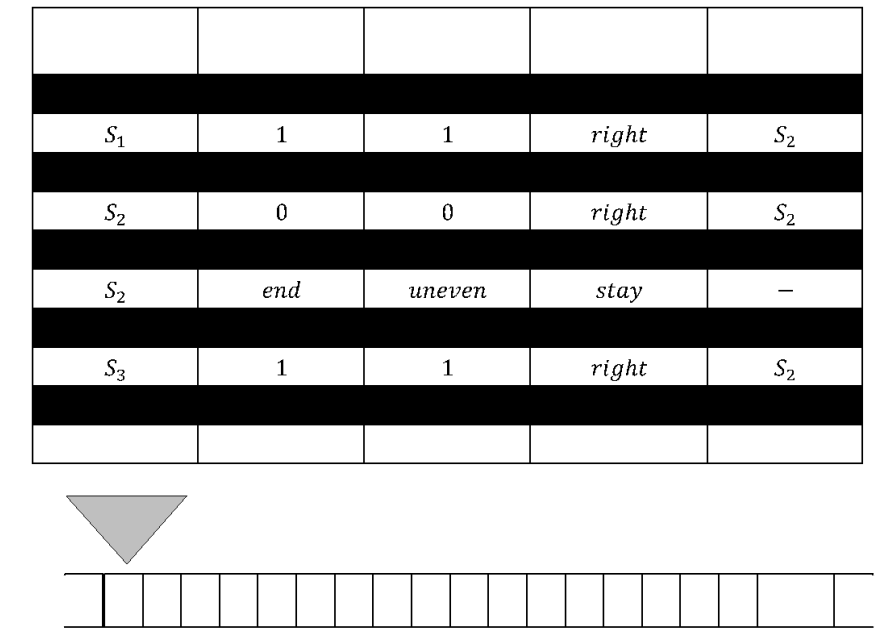

Figure 1. The Turing machine consisted of an infinite squared paper tape, a reading/writing head and rules that controlled its execution. The rules in the picture describe an algorithm to check if the number of ones is odd or even.

The rules were central in Turing's machine. Each rule had an identifier and different versions depending on the content of the square under the reading/writing head. A rule could control the reading/writing head to write a symbol, move a step to the left, right or stay still on the tape. The rule also defined the next rule. With the rules and the machinery any calculation could be defined unambiguously. One other invention was that a group of rules could have common identifier which allowed for modularity were already defined calculation could be reused. Turing used this feature to define a machine that could simulate any other Turing Machine (TM) if its description were written on the imaginary paper tape. The universal machine allowed Turing to prove that a machine that could check if the other machine produced a result was impossible. This meant that there is no general procedure by which one could calculate if the formula is valid.

When Turing was about to publish his findings in 1936, he discovered that the American Alonzo Church had beat him to it. Church used different kind of computational model, which was based on functions and simplification. Turing's article was published anyway because of the uniqueness of his method, but he published later an addendum where he acknowledged that Church was first. In the addendum he also described a TM that implemented Church's computational model. Both models were equal in computational capacity. Around the same time several other computational models were invented which differed in the way the computation was executed. They were all equal in capacity and the thesis is that they defined the limits of what can be calculated.

Models of computation preceded the invention of the modern computer, but many of the ideas that are in use today can be said to have been thought of in relation to formalizing calculation in mathematics (Davis, 2015). The TM for instance incorporated the idea of a programming language (rules), modularity, program and data in the same memory, virtual machine and interpreter. Turing came up also with the idea of distributed computing where a Turing machine could receive a part of the solution from an outside source. The Turing machine was a mechanization of how a human calculates, so it wasn't only theoretically but also intuitively a model of computation. A sign of importance of Alan Turing work is the A. M. Turing Award issued by the Association for Computing Machinery (ACM) which is one of the two leading CS associations in the world. The Turing machine is still used for formal proofs in theoretical CS.

Besides computation mathematics provides CS with the mathematical systems that are the objects of computations. One could therefore think of CS as a branch of mathematics (Tedre, 2018). When it comes to researching properties of formal systems and using these formalisms to investigate other systems it is without a doubt mathematics. However, when it comes to implementing computational models in forms of computer machines there is also the question of physical implementation. What started as electrical engineering evolved to the independent fields of computer engineering which built the technology and software engineering which defined how to control it. Computer science combines theory building with practical application.

Engineering

Besides mathematics also engineering is central to CS. The first computers were built in universities and from that practical work the science was formed. Engineering can be defined generally to be: "…discipline of using scientific and technical knowledge to imagine, design, create, make, operate, maintain, and dismantle complex devices, machines, structures, systems, and processes that support human endeavour." (Blockley, 2012, p. xi). The first computer was built to provide more computing capacity for calculating gun firing-tables during World War II in United States. Every new gun needed a table of approx. 3000 values, and it took one month to calculate the numbers for a hundred human computers. It took the same time with a human operated mechanical differential analyser which there were at the time three of in whole USA. The army collaborated with Moore School of Electrical Engineering in producing the tables and they had together two analysers and a two hundred human computers. Still, they couldn't produce the tables fast enough to keep pace with the gun development.

The need for computing capacity created the idea for a fully electronical calculating machine at the Moore School. The army agreed to finance the idea and in 1943 started the project to create the Electronic Numerical Integrator and Computer (ENIAC). The design of ENIAC took inspiration of how calculations was distributed to the human computers and how the differential analyser combined partial results. To avoid slowdown in the operation ENIAC was to be fully electronic. Vacuum tubes were used to present control units, arithmetic circuits and store the numbers ENIAC was operating on. Like current computers ENIAC was designed to be general purpose (mathematically), but it had a distributed design with several calculating units and used decimal representation. The project finished in December 1945. ENIAC was a thousand times faster than the best of the competing calculation methods of that time.

ENIAC was a success both technically and operationally, but programming was very difficult. The length of ENIAC was thirty meters and it consisted of forty units. Each unit was controlled by a switchboard and additionally the different units needed to be connected by pluggable wiring in accordance with the planned steps of the calculation. The team behind ENIAC came up with a design for a new kind of computer in collaboration with the mathematician John von Neumann. In 1945 a preliminary report with only von Neumann's name on it was circulated which described the design for a new computer. The design got the name von Neumann architecture and todays computers still follow it. It describes the computers main components and how they interact with each other, the technical details are up to the implementer (which can be a reason for its longevity).

von Neumann architecture described the design for a machine which consisted of input unit, arithmetic/logic unit, control unit (in modern computers logic/arithmetic and control is in a single central processing unit), memory and output unit. The different units were connected via a parallel bus (see Figure 2). The computer was controlled by commands which resided in the memory with the data. Both commands and data were represented with binary numbers that is a series of zeroes and ones. The use of just zeroes and ones were technically very economical compared to decimal representation. Commands were executed sequentially which concerned both handling of data and control of the computer. On the material level the computer commands are just group of computer components having a certain charge which can affect the operation of other components. Abstractly the commands form a description of the intended

computation based on a computational model. In between the material and the abstract is a liminal artefact, the physical representation of the program to be executed (Dasgupta, 2016).

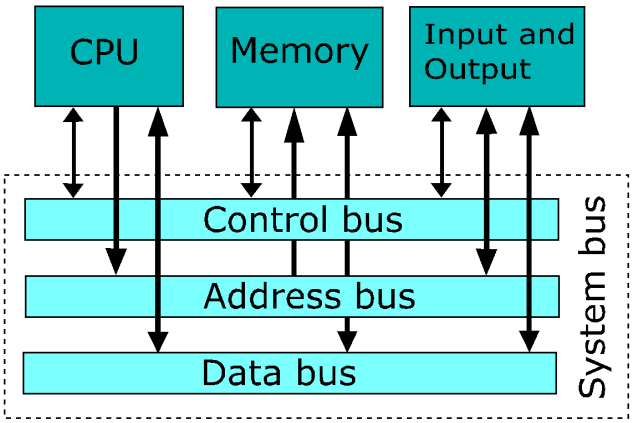

Figure 2. (CC BY-SA 3.0) [1] von Neumann architecture describes the modern computers main components and how they interact with each other. The main components are the input unit, central processing unit (CPU), main memory, and output unit.

The first implementations of von Neumann architecture were in United Kingdom. Mark I was ready in April 1949 at the Manchester University and only a month later EDSAC (Electronic Delay Storage Automatic Calculator) was finished at Cambridge. The development of EDSAC was a joint venture between a private company and the university. J. Lyons & Company was Britain's biggest catering company which depended on a huge finance department to keep the company profitable. They were always looking for technical aids to improve the mechanical part of the finance operation and they had heard of the new computer. The deal was that they financed the development of EDSAC and later the university would help them build their own computer. J. Lyons & Co. computer was called LEO (Lyons Electronic Office) and they used it for wages, orders, and shipping management, and for managing the manufacturing of their tea blends. LEO was the first business application of computing and to commemorate this the Association for Information Systems (AIS) gives the LEO Award to outstanding scholars in the field of information systems.

There were possibilities for many kinds of implementations since the von Neumann architecture only described the computer's general organization. At the time, the people who built computers were electrical engineers. It required ingenuity to take inventions developed for other purposes and repurpose them to build computers. Programming was an integral part of von Neumann architecture. It was possible to choose if a feature was implemented in hardware or by programming. Maurice Wilkes which was one of the designers of EDSAC invented microprogramming which was a set of programs which made the job of programmers easier since the programs could be used as higher-level commands to control the computer. The

[1] By W Nowicki - Own work, based on a diagram which seems to in turn be based on page 36 of The Essentials of Computer Organization and Architecture By Linda Null, Julia Lobur, https://books.google.com/books?id=f83XxoBC_8MC&pg=PA36, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15258936

specialization of Computer Engineering (CE) started to emerge in Electrical Engineering (EE) departments at universities in step with the more widespread use of computers and evolution of computer specific technology.

Modern CE is about the design, construction, implementation, and maintenance of software and hardware components of computing systems, computer-controlled equipment, and networks of intelligent devices. CE started as a combination of EE and CS. It has evolved over the past four decades as an independent discipline, but EE and CS are still its reference disciplines. Even though CE started with designing core components of computers very few engineers are today involved with these. Through the miniaturization of computer technology this has become highly specialized area. The miniaturization of silicon devices, increased reliability and complete systems on chip have made computers ubiquitous and replaced many conventional electronic devices. Computer engineers are needed in making of smart phones, tablets, wireless networks and other digital products. Through embedded systems they also work with automobiles, household appliances, consumer electronics and lately developing the internet of things. While ACM was more for the theoretical computer scientists Institute of Electrical and Electronics Engineers (IEEE) Computer society is more for the technology-oriented researchers of computing.

Programming controls the operation of the computer and without the programs' computers wouldn't function. Computers were created for mathematical calculations, but in parallel with their physical development, hardware, also the programs running on the computer, software, evolved. Other uses for software were invented, and these made the programs grow. Mathematical methods weren't enough to handle the complexity of programming and this phenomenon came to be called the software crisis. NATO arranged a conference in 1968 and another 1969 about how to manage software complexity where methods from engineering sciences were proposed. This was a start for a new field called Software Engineering (SE). A modern definition of SE is given by the IEEE: "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software." (IEEE, 2010).

Software engineering is about methods, tools, and management of software development. The main challenge is complexity: how to structure the program code to make development more manageable, allow several people to work at the product at the same time and how to coordinate this activity. Procedures and functions were the first methods to structure code, ideally the first one changes the computers state and the other one returns a result; both can be called from other parts of the program. Object-oriented programming is a way to build the program from independent components. Each object can have several methods which is an abstraction of procedures and functions that is they can behave either way. Design patterns took inspiration from architecture to describe a general pattern of interacting objects that solve a particular problem. In general, architecture is also used as a term to describe the overall principle of how the software is structured.

By making software more modular the task of programming can be divided between several people. It is important that the behaviour of a component is agreed on since other parts of the software may rely on it. This is also a source of error when changes are made to an existing part of the program. Modern development environments allow to search for dependencies between different parts of the software. They also help in showing the general structure of the software

and highlighting the different classes of commands the program consists of. Modern software projects are large, and several people are needed to develop them. Version control system help to manage the different parts and versions of the software in development. There are two major strategies to manage the software development project. One is called the waterfall method where the development is progressing in stages of requirements, design, development, verification, and maintenance. The other type of development is called agile where the development progresses in iterations and features are added to the whole in each iteration. It is said that waterfall method is preferable in larger projects and agile method is better when the project is small to medium or when there is uncertainty of the final feature set.

The engineering view is about the technical building of both computer hardware and software. Our current digital society is the result of technical development from the start of the computer era. This development has happened in universities, but also in many companies whose products have made the digital revolution possible. Theoretical work has been important part of the development. Especially in the beginning many things wouldn't have been possible if not theoretical methods had been developed to get the most out of existing technologies. When the principle was invented then it was possible develop it further. Both the theoretical and technical started from the same origin modelling of human calculation. With this modelling came the idea of simulating human intelligence. Here computing is used as a theory of intelligence and its example of the scientific perspective of CS.

Science

The science perspective is about CS as empirical research like in the natural sciences where there are objective phenomena that is investigated. It is distinct from the mathematical perspective which concerns the theoretical aspect of CS and engineering which is about building hardware and software. Science can be described by the techniques of experimentation, observation and theory construction (Okasha, 2016). Not all fields can use experimentation e.g. astronomy, but there needs to be a way to gather data to validate the theories. In 1956 a summer school in Dartmouth, USA was held for researchers interested in a new topic of CS, Artificial intelligence (AI). The research of AI was based on the idea that: "… every aspect of learning or feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." (McCarthy et al., 2006, p. 2). Here CS is used in a scientific way to describe an outside phenomenon.

AI allowed CS to participate in the interdisciplinary research of the mind named Cognitive science. The mind is a complex topic that combines research areas of many sciences: philosophy, psychology, linguistics, anthropology, neuroscience, and computer science. The theoretical model of computation and the computer gave the other sciences tools to model and simulate what happens in the mind. It can be said that Cognitive science was born the same year as AI in a seminar at MIT (Miller, 2003). The famous linguist Noam Chomsky described what kind of capacity is needed to produce language. He compared Turing Machine (TM) with machines of lesser capability to show that to produce language a system equal to TM is needed. Georg Miller showed that a human can handle a maximum of seven things at a time in working memory, but through grouping the amount of information can be raised. Allen Newell and Herbert Simon introduced Logic theorist (also presented at Dartmouth) which was a program

that could prove mathematical theorems. Logic theorist used heuristic algorithms which simulated how a human reason's when faced with uncertainty.

Another important application of CS to the study of the mind was based on the idea that neurons of the brain could be interpreted as performing logical functions by Warren S. McCullough and Walter Pitts (Bermúdez, 2020). The brain has 86 billion neurons, each neuron can have over 10 000 connections and together they form a large network. We know that the neurons communicate by electric signals and this activity can be monitored. Neuroscience has been able to map activity in different brain areas to cognitive functions. But what the activity means, that is what kind of representation (assumedly) is used, is still unknown. McCulloch and Pitts work gives a mechanism where artificial neurons operate on values of 0 and 1 like a computer. This allows a computer to simulate the operations of a neural network. Artificial neural networks have made it possible to model the operation of the brain, simulate cognitive functions and make intelligent software. It is the existence of target phenomena and related research results to compare with that decides if a simulation is science or engineering.

CS can be described as the study of natural and artificial information processes (Denning, 2007). Information Systems (IS) as research of business or in general organisational use of computers combine both views. An organisation can be interpreted as a system of interacting processes that produces the products or services to fulfil its purpose (Checkland & Holwell, 1998). Every process needs information to operate and produces information that other processes may rely on. In addition, information is needed to manage and coordinate the system. Information systems was born out of the need to plan and design the use of information technology to support the information needs of the organisation. This is the design side of IS, but there is also a behavioural one whose reference science is sociology (Hevner et al., 2004). IS as a science is research about how people create and use information to enable the operation of the organisation. Also, how information technology impacts organisational functions is another area of research.

AI and IS both contain research that relate to outside phenomena and where empirical methods are used. But there is also an idea that the combination of theoretical and engineering perspectives of CS could be seen as empirical science. To qualify the research should be rigorous and the methods used should equal those in natural sciences. From the literature one can find five major views on experimental CS (Tedre & Moisseinen, 2014): feasibility, trial, field, comparison, and controlled experiment.

Feasibility Experiment concerns new techniques and tools. The aim is to explore if a task can be automated cost-efficiently, efficiently, feasibly, reliably, or by meeting some other simple criteria. The method of the experiment is a demonstration that it can be done. A *Trial Experiment* goes further than feasibility and aims to evaluate if a system confirms to its specification or how well it performs. Its performance is evaluated against a set of predefined variables. The experiment is usually laboratory based but can be also conducted in an actual context if one takes into account the lesser control of the environment. The last option is also the setting for a *Field Experiment* where the aim is to test the system in use. Here the contingencies of the real environment are part of the experiment. The aim is to test the performance, usability, or robustness of the system. These kinds of experiments are used often in the field of information systems where success is measured by benefits in practice.

In many branches of CS, one is looking for solution with the best performance. It can concern use of resources, speed, completeness, quality of service or some other factor where one can judge if a result is better or worse. These are called *Comparison Experiments* where the purpose is to evaluate if one solution is better than the another when the experiments are made under the same conditions. *Controlled experiment* can be said to be the gold standard of science and it can also be used in CS context. It means that care is taken to control all the affecting factors. The simplest type has positive and negative controls, which are usually sufficient to eliminate uncertainty with variables. The controls mean that negative and positive results are produced when expected.

CS and the ubiquity of computing technology has also changed sciences like other parts of the society. The Nobel laureate Wilson (1989) wrote about the use of computers for science which he called Computational Science. He recognized that there are alongside the traditional experimental and theoretical research now a third method computational research. However also experimental and theoretical methods have changed because of the possibilities of modern computing (Denning, 2017a). Experimental science is about collecting data through observations and experiments to confirm or reject a hypothesis. Computers allows the handling and analysis of large data sets. Since the availability of computer resources have grown massively in the recent years the scientist doesn't have to settle anymore for statistical analyses of samples but can analyse the whole data set. In theoretical science the scientist builds mathematical models to explain what is known and uses the models to make hypothesis of possible phenomena. Computers help in calculating the equations the models are based on. The use of computers has not changed the fact that both data and models (theory) are needed to make science.

Using computers to accelerate experimental and theoretical research is a revolution on its own (Denning, 2017a). However, computation has more potential. Instead of static models one can build dynamic simulations that enact the processes that are described. This allows for exploring things that can't be observed or allows to see the consequences of complex interacting systems. These systems couldn't be calculated since there are too many variables for equations, but one can build the components and let the computer handle the interaction. In AI computation was used to simulate human information processing. It seems that also other natural processes can be interpreted as information processes (Denning, 2017a). Information processes receive input and produce output. A computer is particularly suitable tool for calculating these kinds of results. CS is the scientific research of computer technology; it can be used to understand the workings of human intelligence and organisations and it is a great tool for the other sciences. However, to be a science the scientific method must be followed even if it is tailored to the characteristics of CS.

Summary

There is clear evidence for the three perspectives of CS. Mathematics provide the theory of computing by giving an explicit and rigorous definition of an algorithm that is the bases of all computations (see Table 1.1. second column from the left). This allows to explore what problems can be solved by computation using mathematics. Computers expand the possibilities of mathematics by allowing handling of large quantities of data and big complexity that would not be feasible to do manually. Our digital society is the result of development of information

technology by computer and software engineering. The engineering side of computing is constantly finding new application areas by exploring what tasks can be accomplished by controlling information technology (see Table 1.1. third column from the left). Information technology is used scientifically to simulate human intelligence and as a model for information processing in organisations. Many scientific phenomena are too big to be understood by traditional means. The mechanisms that lead to some phenomena are not always observable and here computers can be used to simulate the possible causes. Scientifically computational processes can be used to understand phenomena that without computing wouldn't be accessible (see Table 1.1. fourth column from the left).

Table 1.1. Three perspectives of Computer Science

	Mathematics	Engineering	Science
Object	models of computation	computer and software technology	natural and artificial information processes
Method	calculations and proofs	electronics, computer technology and programming	empirical research, computational models, simulations
Knowledge	possibilities/limits of computation	how to design and produce both hardware and software	computational interpretation of phenomena
Question	What problems can be solved by computation?	What tasks can be accomplished by controlling information technology?	What phenomena can be understood as computational processes?

Lecture - Introducing the three perspectives of computer science

Using the text above

- Mathematics: what problems can be solved by computation
- Engineering: what tasks can be accomplished by controlling information technology
- Science: what phenomena can be understood as computational processes

Activity 1.1 Homework - Looking at own topic from one of the computer science perspectives

Write a short essay (200 words) based on an internet search or/and auxiliary material provided by the teacher describing how an example related to your own subject utilises computer science. You can use the CS perspectives as an aid in your search (see above Table 1.1).

Unit 2 – CT Perspectives for STEAM framework

Here the perspectives of computer science are combined with computational thinking to design computational support for STE(A)M education. The three perspectives where mathematics, engineering, and science, which all answer to different type of question.

Questions answered by the three CS perspectives:

- Mathematics: What problems can be solved by computation?
- Engineering: What tasks can be accomplished by controlling information technology?
- Science: What phenomena can be understood as computational processes?

The PRADA model (Dong et al., 2019) for computational thinking was devised to be a practical way to integrate computational thinking in K-12 education. The aim was to give an understandable picture what computational thinking is. The PRADA acronym consists of Pattern Recognition, Abstraction, Decomposition and Algorithms. Here we elaborate on the CT concepts of PRADA and see how the CS perspectives affect the interpretation.

> Observe! We use the PRADA model that emphasizes the conceptual part of CT to aid the preservice teachers in choosing a suitable computational tool for teaching their target subject. It differs from the CT model introduced in Module 2 Unit 1 (Erik Barendsen) which has a stronger problem-solving emphasis.

PRADA model:

- **Pattern Recognition** observing and identifying patterns, trends, and regularities in data, processes, or problems
- **Abstraction** identifying the general principles and properties that are important and relevant to the problem
- **Decomposition** breaking down data, processes, or problems into meaningful smaller, manageable parts
- **Algorithms** developing step by step instructions for solving [a problem] and similar problems

Pattern recognition

Patterns are here interpreted in two ways mathematical and technical. Mathematics is said to be "the queen of sciences" because it provides the concepts that are used to form theories. The technical interpretation of pattern covers both engineering and technology. The difference is in the stage of development from design to product. Mathematical patterns (Devlin, 1994; Kvasz, 2019) describes regularities in a field of mathematics which can be defined with a small set of

facts and rules. From these facts and rules one can derive logically all the other concepts of the field (Devlin, 2012). Here we use patterns to make explicit the often very abstract mathematical concepts we are using in our computations. Technically in the context of computational thinking patterns are about controlling computers, electronics, and embedded systems. The requirement is that the operation of the target system can be described with discrete electric states and transitions between them. In both mathematical and technical views, the first step towards a solution is to recognize the patterns that frame the problem.

Learning mathematics can be seen as an analogy of learning a language (Sfard, 2007). Without the language, we wouldn't be able to access the abstract objects of mathematics and their properties. Keywords in mathematical discourse signify the objects. For instance, we use numbers to stand for quantity and shapes for geometric objects. The keywords also describe relationships between the objects like equality, inequality, similarity, equivalence, etc. Every subfield of mathematics has their own keywords, but one field can use the keywords and objects of another if they are part of its constitution. When we talk about mathematical objects, we use words, but for visual communication we use a system of symbols which are much more succinct and effective. The system of symbols includes digits, algebraic and logical notations, and formulas. The symbols function also as cognitive aids. We can also visualize the structures that the objects and their relations form like geometric drawings and graphs. Diagrams make possible to show visually the results of formulas and functions with different inputs.

Every subfield of mathematics is described by a system of facts and rules that define its objects. A new mathematical definition must be proven to be deducible from the system. A proof is a narrative about the objects and the relationships between them that form a definition. The narrative is accepted or rejected by mathematicians of the subfield depending how well it adheres to the established understanding of the mathematical system in question. Sometimes the existing system can be changed if the new definition is of a fundamental kind, but these occasions are rare. The narratives use axioms, definitions, claims and proofs to establish new findings. The language of mathematics is formal and rests on a routinized way of talking about objects of interest. One important type of routine are the actions on the mathematical objects. Examples of these routines are calculating, problem solving, validating, and proving.

The mathematical objects and their relationships can be said to form a pattern. The pattern is a basis for our calculations and gives us the vocabulary to discuss about the mathematical field. Usually, the pattern is not the core of the mathematical system, but a consequence of it. We have been taught the concepts and their relationship in mathematics lessons or books and practiced them by doing calculations. Given a problem we can use the patterns to recognize if a subfield of mathematics would help in solving it. Here we use mathematical theory of numbers better known as arithmetic (see Figure 2.1). to illustrate a mathematical pattern.

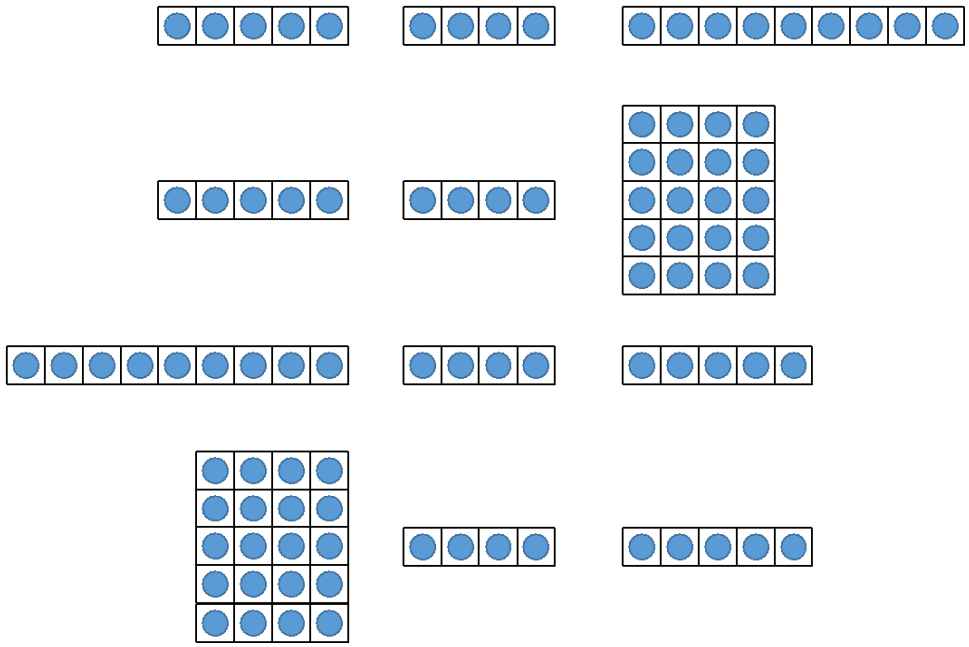

Figure 2.1. Addition, multiplication, subtraction, and division of arithmetic can be visualized to show quantitative patterns related to the symbolic operations.

The American National Research Council's Framework for K-12 Science Education (NRC, 2012) lists developing models, data analysis and, mathematical and computational thinking among the eight central scientific and engineer practices. The framework is repeated in the American Next Generation Science Standards (NGSS Lead States, 2013) which combines the practices so that mathematical representations (models) are used to support scientific conclusions and digital tools (computational thinking) for analysis of large data sets. Also, frameworks like Computational thinking in mathematics and science taxonomy (Weintrop et al., 2016) and Elements of Computational Thinking Integration from a Disciplinary Perspective (Malyn-Smith et al., 2018) emphasize the use of models and data as important part of scientific practice. Modelling and data analysis can be interpreted as recognizing patterns.

The most common meaning of a model is a representation of an idea, object, event, process, or system that is created for a specific purpose (Gilbert and Boulter, 1998). The basic characteristics of a model are:

- It represents some part of the system being modelled and the aspects chosen reflects the interest of the modeller. A model is a human creation, and it doesn't exist in the natural world.
- The same reality can be represented by multiple models, depending on the features that are interesting to whomever is creating the model.

In science a model can be seen as a representation of a system, which is made up of set of objects and their properties or variables (Gutiérrez & Pintó, 2005). The model depicts the laws of the system that defines the behaviour of the objects or the relationships between object variables. The essential function of a model is to explain and predict. Scientific models are also

representations to reason with (Justi & Gilbert, 2002). Models are often mathematical, but they are on higher abstraction level than plain mathematical concepts. Instead of a mathematical subfield's basic objects and operations the models combine them in formulas to describe the phenomena of interest. A mathematical model can be thought of as an imaginary and simplified version of the part of reality being studied where exact calculations are possible (Gowers, 2002). Newtonian mechanics is an example of this kind of model (see Figure 2.2).

1) $\sum F = 0 \Leftrightarrow \frac{dv}{dt}$ 2) $F = \frac{dp}{dt} = \frac{d(mv)}{dt}$ 3) $F_A = -F_B$

Figure 2.2. Newtonian mechanics is another name for classical mechanics in physics. Isaac Newton laid the foundation for classical mechanics by defining the three laws of motion: First law defines that an object stays still or continues it movement if not affected by an outside force (1), Second law defines that the sum of the forces affecting an object is the mass of the object multiplied by its acceleration (2) and Third law defines that when one object exerts a force on another the second object exerts an equal force on the first one. These laws are examples of scientific patterns in physics.

The previous picture of science is based on the idea that a description of a phenomena can be simplified to a degree that an exact calculation can be used to extract information from identified variables. Not all phenomena can be described this neatly for instance the common features of some human population, weather trends, economic forecasts or the spread of disease include variations that have to be accounted for. Instead of discrete and exact values we have large amounts of data where the objects in a category can have different values for the same variables. All phenomena are not static so the collected values can vary depending on the moment of measurement. Data can also be used for predictions and besides the prediction itself it is informative to estimate how certain or probable the result is. Statistical models are used to handle variation and uncertainty in data. The aim is to get an understanding of what the data means or entails.

Statistical inquiry can be said to be a process containing of five phases: problem, planning, data, analysis and conclusion (Wolff et al., 2016; Wild & Pfannkuch, 1999). First the purpose of the inquiry is identified and what are the relevant questions related to it. The planning starts with hypothesis formulation, that is what kind of results are assumed. Then a plan is made that describes what kind of data could confirm or reject the hypothesis. Also, the possible sources of data and how they can be obtained are listed. In the data-phase the data is collected or acquired. Here one must take into account the quality of the data and possible ethical concerns like consent, anonymization and different kind of permissions. One of the central phases in the process is data analysis where the results are gathered, and explanations based on them are created. Finally, the validity of the explanation is evaluated, and possible new questions based on the results are created. There are many methods and tools to base the data analysis on, here we use the common statistical measures like arithmetic mean and standard deviation as examples (see Figure 2.3).

1) $A = \frac{1}{n}\sum_{i=0}^{n} a_i = \frac{a_1+a_2+\cdots+a_n}{n}$ 2) $\sigma = \sqrt{\frac{\Sigma(x_i-\mu)^2}{N}}$

Figure 2.3. Basic statistical measures like Arithmetic mean and standard deviation give summaries of the patterns found in numerical data. Arithmetic mean describes the average value of an attribute in a data set (1). Standard deviation describes how the much the values of the attributes differ from the average in the data set (2).

Different branches of engineering utilize the patterns of mathematics and science in their work. In the context of computational thinking there are also technical patterns related to control and automation of electrical, electronical, and computational devices. To control an electrical device computationally it has to be a computer, or it has to have the electronics to communicate with one. In simple cases the device could use electronics for automation and control, but for more complex needs it is easier to add computational capacity than to try to build it with just electronic components. Computers and electronics work in discrete states which means that some electrical charge or current is either on or off. This doesn't mean that the device has to be static. A state can be that a motor is running or even accelerating. A model can be made to describe the states controlling or automating a device.

A model of control or automating mechanism of a device needs a start state, transitions between states, inputs that controls the transitions. This kind of model is called a state machine or a finite state machine (FSM) since the number of states are limited (see Figure 2.4). The FSM is simplest computational model that have a memory. Combinational logic is simpler since it can just calculate a result, but because it lacks memory it can't handle states. An FSM changes states by transitioning to a next state when it receives an input. Usually depending on the state of the machine the number of possible transitions is limited, and different states can have different transitions. FSM is defined by the list of states, the initial state and the inputs that trigger transitions. Even though the modelling formalism is simple a wide variety of devices can be modelled. The FSM is an abstraction, it describes the results of the operations of the device, not the actual mechanics.

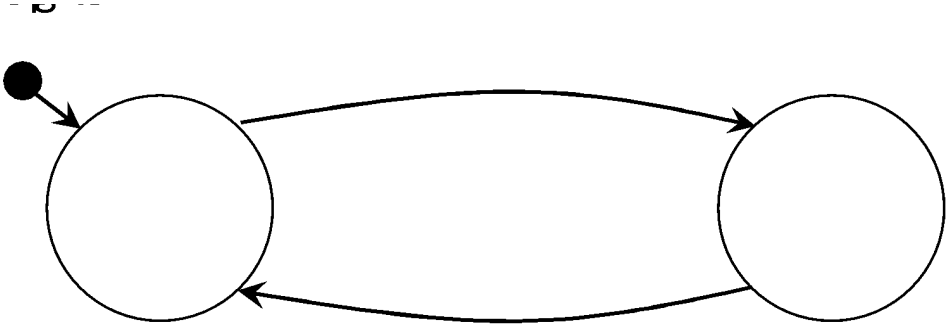

Figure 2.4. A simple Finite State Machine (FSM) describing a system consisting of a switch and a light bulb. The FSM describes the states of the light bulb while the switch changes states. Flipping the switch up turns the light on and flipping the switch down turns the light off. The default or start state is light off and the switch is down.

Models of FSM can get very complex if there are many inputs and states since every combination have to be modelled. Statecharts where devised by Harel (1987) to make

modelling of a FSM easier (see Figure 2.5). Statecharts employ different graphical means to make a state machine model easier to interpret: clusters, default-states, and-states, and history-connector (Thimbleby, 2007). If an input triggers a state transition to a group of states that are complementary, then these can visually be clustered inside a node. The state which is always first in a group of states can be marked as a default-state. If several states have something in common, but differs in some aspect, then this can be modelled with an and-state. Visually an and-state is a cluster that is divided in two which allows a combination of substates. Sometimes the last state in a state cluster needs to be remembered and this can be marked with a history-connector. If a history-connector is attached to a state cluster then an additional star after the cluster name could mark that it is one of the states inside the cluster that is meant.

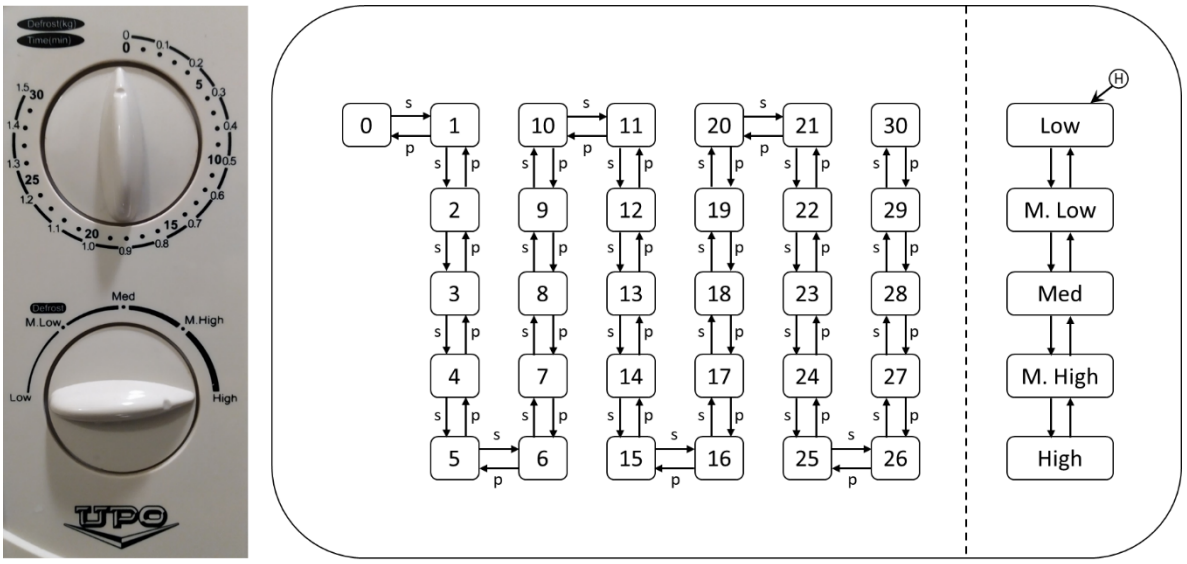

Figure 2.5. A statechart describing the operation of the user interface of a simple microwave oven. The interface consists of a timer dial (0-30) and setting for the power level (low-high). The arrows marked with s in the middle of the picture describe setting the time as changing states and the arrows marked with p the change of time by a time pulse. Using statechart formalism simplifies the description of the combination of two state hierarchies by allowing them to change separately. The power level remembers (H) its last state in the hierarchy.

Abstraction

The patterns described above of mathematics, science, engineering/technology are examples of abstractions. The mathematical objects don't exist in reality but are product of our minds. The philosopher Karl popper distinguished between three levels of human experienced reality the physical world, the world of our thoughts and the non-material creations of our culture. Mathematics is culturally created (Hersh, 2014), it has been developed for millennia and disseminated through teaching and writing. Science uses mathematical and statistical models to make exact description of the phenomena of interest. Since the models are usually a combination of basic operations and is named after the phenomena depicted, we can say that they are on a higher abstraction level than plain mathematics. Engineering/technology used state machines from computer science as an abstraction to describe how a device works. State

machines do not describe how devices operate; they model the results of operations. In general, to abstract something is to leave out details or group details that are related under a descriptive label.

Mathematics, science, and engineering/technology all employ abstractions. Abstraction is also central concept in CS (Kramer, 2007) and therefore also in CT (Wing, 2006). However, CS/CT requires thinking on multiple levels of abstraction, often two levels at the same time. One example of levels of abstraction used in CS education context is a four-level hierarchy by Perrenet et al. (2005): problem, object, program, and execution (see list below). The levels in the hierarchy are describing different interpretations of an algorithm in programming. At the problem level an algorithm is the change required from inputs to outputs. The inputs are based on the patterns that frame the problem. Object level is thinking about the general steps of the algorithm that are needed to transform the inputs to the outputs. On a program level the algorithm is implemented in a programming language. The programming language is a high-level description of the operation of a computer or a computational model. The execution level is the physical operation of the computer. Here we have taken the steps from a pattern that is in the context of the problem area to defining an algorithm to solve the problem and its solution as a program executable on a computer. CT is the ability to perform this process.

Abstraction levels of algorithms Perrenet et al. (2005):

- Problem level: change from inputs to outputs
- Object level: algorithm (high-level) description
- Program level: implementation of the algorithm in a programming language
- Execution level: the running of the machine

Decomposition

Decomposition is about how to break down a problem into manageable parts. In CT context the aim is to find an algorithm that would solve the problem. Before decomposition, a problem is only known as a black box with inputs, outputs, and possible relationships to other problems. The black box depicts the innerworkings of the problem that are yet unknown. To find out the inner function, the problem is broken down into multiple subproblems. Each subproblem can be treated as a problem that needs to be unpacked to its component parts. Finally, an atomic level is reached where there are no more subproblems. A solution can be devised based on the identified components of the problem. One can approach the solution top-down, first handling the top-level problems, or bottom-up, first solving the atomic problems.

The decomposition process contains two kind of actions (Rich et al., 2019). The first, substantive decomposition, is to differentiate and categorize the parts (subproblems) of the problem. The second, relational decomposition, is to find relationships between the subproblems. In substantive decomposition one must choose a principle that is used to break the problem in to parts. The principle is dependent on the problem statement and context. The separate subproblems provide information that wasn't accessible while combined. When subproblems have been identified one can start to assign relations between them. Prior to this phase there didn't exist a relation between the problems. Subproblems and their relationships form the pattern of the problem. An example from arithmetic would be the addition of two natural numbers. The relationship between the numbers is the difference between positions on a

number line. To add one number to another one would take the first number's position as a start and calculate an amount equal to the second numbers position forward. The number of the new position is the result of the addition.

In engineering, CS, design, and other fields, problems are frequently broken down by their functions (Rich et al., 2019). Functional decomposition is a result of the combination of substantive and relational processes. The key to make the decomposition functional is the relationship between subproblems. If the relationship is functional then some operation happens between two subproblems. A functional relationship in arithmetic could be addition, subtraction, multiplication or division. Example of the use of arithmetic functions in other fields of mathematics are the quadratic equation, the greatest common divisor and the Pythagorean Theorem. In CS an algorithm expressed in an imperative type (commanding the computer) of programming language would employ control structures that form the relationship between solutions to subproblems. Breaking down the problem using substantial and relational decomposition helps to gain new meaningful information to devise a (algorithmic) solution (Rich et al., 2019).

Algorithms

Algorithms are part of mathematics. Sometimes any step-by-step process is described as an algorithm, but it is not the process, but the definition that is the algorithm. The definition has to describe the process with mathematical rigour. CS relies on algorithms since its operating principle is based on a computational model (see Unit 1/Mathematics). Everything that the computer does is a calculation (called computation), but for the machine to know what to do every action has to be explicit. In a CS context an algorithm can be defined as *"...a finite, abstract, effective, compound control structure, imperatively given, accomplishing a given purpose under given provisions."* (Hill, 2016, p. 47).

To decompose Robin K. Hill's (2016) definition of an algorithm (see above) one can start by picturing an algorithm as a definition of a process consisting of discrete steps. An algorithm describes a finite process that has to come to an end. The description is abstract and uses only the features needed to depict the change intended. In addition to finiteness the algorithm has to be effective which means that the resources and time used has to be reasonable. An algorithm is a combination of different control structures that guide the change of its values and finally ends in a result. Control structures and changes to the variables are presented as imperatives using commands. The algorithm should fulfil its intended purpose and nothing else. The algorithm should behave correctly with any input in the specified range.

Computational thinking can be defined as the mental skills needed to automate the execution of an algorithm (Denning, 2017b). The properties of the algorithm defined above are a prerequisite that it can be implemented, it behaves as expected and that the result is correct. To implement is to write a program that the algorithm is a part of. Even though every program is algorithmic only those parts of the program that are general enough so that they could be reused in some other contexts are called algorithms. These general algorithms are named, and their definitions are shared among computer scientists. Lots of program code has to do with the control of the computer in a specific context and the code is not usable in another. A program is written in relation to the properties of the computer. One could control the computer directly,

but this is very difficult and error prone. In general programmers' work with a higher-level computational model which is described by a programming language.

A program is written in a programming language. The grammar of the programming language is called a syntax. The semantics define the meaning of the sentences in the language. The meaning can refer to calculations or the functioning of the computer. A program is a list of sentences which are called statements. The statements are executed one at a time in given order. A statement consists of a command and values that the commands need for its execution. Values can be defined directly, but usually some letter symbol is used as a stand in like in school algebra. The symbol is called a variable if it can be changed and value if it is constant once set. Sometimes a statement contains mathematical or logical expressions that calculate new values based on the variables/values. The difference between a command and an expression is that the former controls the functioning of the computer and the latter produce values. The result of an expression are inputs to the command. Expressions consist of operators (like addition or subtraction mathematics and AND or OR in logic) that operate on the values/variables. A program execution ends at the last statement if there isn't a statement that would tell otherwise.

The program is written one statement on each line. The execution of the program goes through each line of the program until it reaches the end. Control statements alter how the execution progresses. Conditional statements allow to choose between alternative path of execution. It is possible that the execution returns to the main path after the alternative statements have been executed. A loop statement is a construct that repeats a set of statements until a condition is met. After the execution of the loop has ended the program execution continues if there are statements left. Programs can get quite large which makes it hard to understand the program code. Sometimes the functionality of a part of a program would be useful in another. Functions and procedures provide modularity and reuse. They are subprograms that can be named and called like existing commands of the language. The difference between a function and a procedure is that the first one returns a value and the second changes the state of the computer. Python and Racket are examples of textual programming languages used in education. There are also visual programming languages aimed at novices where programming resembles building a puzzle.

A puzzle-based approach to visual programming is also called block-based programming (Weintrop, 2019). In this type of programming, the commands of the language are presented as visual blocks. The form of the blocks gives clues on how and where they can be used. The target group of these kind of programming languages can be kids as young as five years, but mostly the languages are aimed at eight- to sixteen-year-olds. To write a program is to combine program blocks by dragging and dropping. Usually, the programming environment prevents the combination of blocks that don't fit together. This allows the writing of a program statement by statement as in textual programming languages while preventing syntax errors. Since programming is made by drag and drop the programming environment can offer additional support by grouping the blocks by function and use colours for easy identification. Knowing or remembering what is possible is replaced by browsing commands.

A side effect of block-based programming is that it relieves the user from writing. The user doesn't have to worry about spelling mistakes, strange punctuations marks or missing pieces of syntax. The graphical representation of statements allows the use of longer descriptions since

commands don't have to be written. It also gives the possible to describe the meaning since the exactness needed for the computer can be hidden from the user. In addition to the block-based language also other aids can be added to the programming environment. In Scratch (Maloney et. al., 2010), which is currently (in 2020) the most prominent block-based programming language, the control of graphical elements on the screen called sprites has been made particularly easy. Each sprite has its own program that controls its behaviour. One of the most important features of the sprite is its location. This makes it easy to make things happen on the screen. Since sprites are independent wholes like functions or procedures in other languages, they can easily be moved from one program to another.

Case study: Modelling gravity in Scratch

Lopez and Hernandez (2015) created an example of a physics project called "Free fall" that could be an assignment for primary or secondary school pupils. The project was implemented in Scratch and it modelled how gravity caused acceleration when an item is falling freely. The physical phenomenon was visualized with picture of a tree and an apple that falls from one of its branches. The velocity of the falling apple is shown as a value and the acceleration is animated by printing the position of the apple in growing increments. To complete the assignment a pupil needs to find out about gravity and what kind of formula describes it effects on a falling item. Using the formula, it would be straight forward to calculate the speed at the end of a distance from an imaginary branch to the ground. The challenge is to use the properties of Scratch to show the increasing speed due to acceleration.

One could think that for pupils in the primary school the result of the program would be shown, but for secondary pupils' part of the assignment would be to come up with the principle of visualization themselves. From the model implementation by Lopez and Hernandez a start would be to add the stage with a three, draw a sprite depicting an apple and move the apple to the correct position by trial and error (see figure 2.6 right hand side). Then comes the task of figuring out how to connect the model of acceleration to the change in a position for the apple. The change in position against time (seconds) would be the velocity and acceleration would be to print the new position of the apple in equal time increments. The start velocity for the apple would be zero (see figure 2.6 centre the first set of blocks).

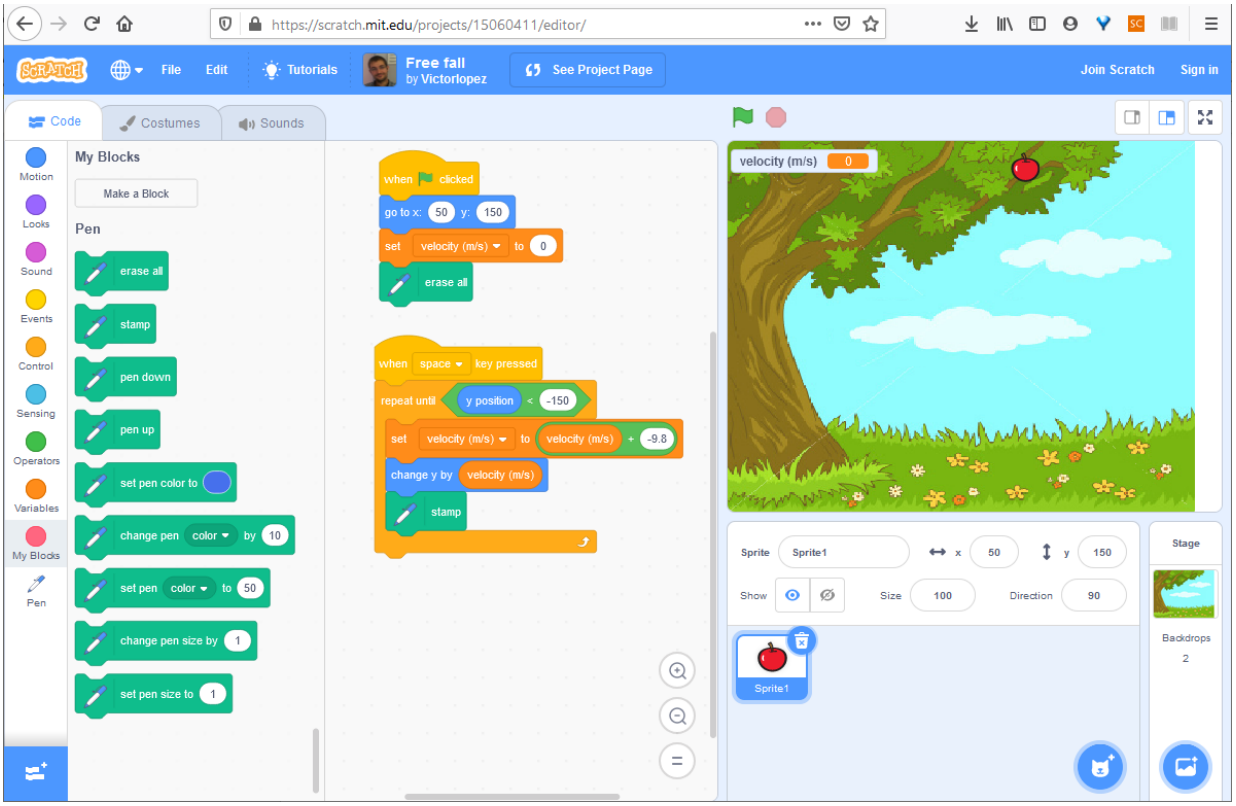

Figure 2.6. A screenshot of the Scratch environment which has the graphical elements of apple in a tree and associated program to visualize the falling of the apply due to gravity.

In the model application a second is depicted as one round in a loop (here named repeat until a condition; see previously figure 2.6: centre the second set of blocks). Now velocity is the change in the position of the apple during one loop. First the effect of acceleration is accounted for by adding 9.8 to velocity (set velocity (m/s)…) and the vertical position of apple is changed (change y by velocity (m/s); see figure 2.6: centre the second set of blocks). Stamp-operation is used in every round of the loop to print an apple (pen and stamp; see previously figure 2.6: centre the second set of blocks). The loop is terminated when the y position of the apple is at the bottom edge of the stage. Once the model has been built there is a possibility to improve the model to be more realistic. Pupils could add air friction or model the effect of wind.

More information about the case study: Scratch as a computational modelling tool for teaching physics (see references Lopez & Hernandez, 2015).

The implementation in Scratch (visited 30.8.2020): https://scratch.mit.edu/projects/15060411/

Summary

The components of CT pattern recognition, abstraction, decomposition, and algorithms as computer programs are all employed in previous example. These are skills that pupils need to know how to utilize the possibilities of computing in STEM. The different components of CT

must be introduced to the pupils: first separately and then combined to form a whole. It is important that programming is not presented only as an elaborate calculator, but that real possibilities of computing are explained. In the example above pupils familiarized themselves with the topic of gravity, built a computational model and used the properties of Scratch environment to visualize it. Depending on the pupils' maturity theoretical explanations of CT can be presented, but without practice computational thinking can't be learned. Also, the teacher needs to practice programming to be able to convey the insights required to create a program.

Lecture - CT perspectives for STEAM framework

Using the text above

- Three CS perspectives
- **P**attern **R**ecognition
- **A**bstraction
- **D**ecomposition
- **A**lgorithm executed by a computer

Activity 2.1 Homework - description of own topic using CT Perspectives for STEAM

The aim of this exercise is to practice applying computational thinking to a topic in your subject. The Modelling gravity in Scratch -case study (see above) is an example application where gravity of physics has been modelled and visualised computationally. You should identify how computational thinking could aid solving a problem, creating a technology or understanding a phenomenon.

1. Choose an interesting topic in your subject.

2. Which CS perspective is related to your topic?
   - Mathematics: What problems can be solved by computation?
   - Engineering: What tasks can be accomplished by controlling information technology?
   - Science: What phenomena can be understood as computational processes?

3. Apply the PRADA model to your topic according to the chosen perspective.
   - **P**attern **R**ecognition
   - **A**bstraction
   - **D**ecomposition
   - **A**lgorithm executed by a computer

4. What kind of exercise could be developed from the computational interpretation of the topic?
   - State the age/grade of the pupils.
   - Describe the idea for an exercise (the implementation details can be omitted).

Activity 2.2 Homework - Peer reviews

- Review Activity 2.1 answers of three colleagues
- Assessment: Accepted/Failed
- Each point is graded on the scale 0-5 (average has to be 2 or better for acceptance)
  - the presentation of the topic
  - argumentation and clarity
  - utilisation of the unit's content

Unit 3 – Choosing educational technologies based on CT Perspectives

Today's pupils are fluent users of digital technologies, but it doesn't necessarily mean that they understand the principles. They need to practice the use of digital technologies first in a more organized setting to be later capable of applying them more freely. STEAM education is about combining knowledge and skills from different areas to solve a problem. Pupils can benefit of experiences with both general tools and tools aimed at a specific subject. In both cases the teacher can use the CT Perspectives framework to recognize educational technologies that combine the handling of topic specific patterns with computational capabilities. In this unit the framework is positioned in a model of teaching to aid in choosing a tool that fit the intended pedagogy. Examples of educational technologies from the three perspectives are given. Finally, the teacher can decide how much the teaching is affected by the technological possibilities.

The Conversational Framework (see Figure 3.1) describes learning as an iterative dialogue between teacher and pupils (Laurillard, 2002; Laurillard, 2012). There are two levels: the communication level where theories and concepts are exchanged, and the practical level where actions lead to experiences. The levels are connected by both teacher and pupils engaging in adapting theories to practice and reflecting on the theories based on the gained experience. In the discourse the teacher disseminates theories and ideas, and the pupils respond with conceptualisations of what they heard. If misunderstood the teacher can modify the message and the pupils can respond with new interpretations. In many topics there is also a practical dimension. The teacher creates an environment where the pupils can put the theories into practice by performing tasks. Depending on the pupils' responses in the discourse the tasks can be adapted to appropriate knowledge level. The teacher sets goals for the tasks that the students should perform. The pupils adapt their actions in light of their theoretical understanding and given goals. Based on the feedback gained from performing the tasks the pupils reflect on their

understanding and adapt their actions to try again. Based on the pupils' performance the teacher modify how she teaches the next time.

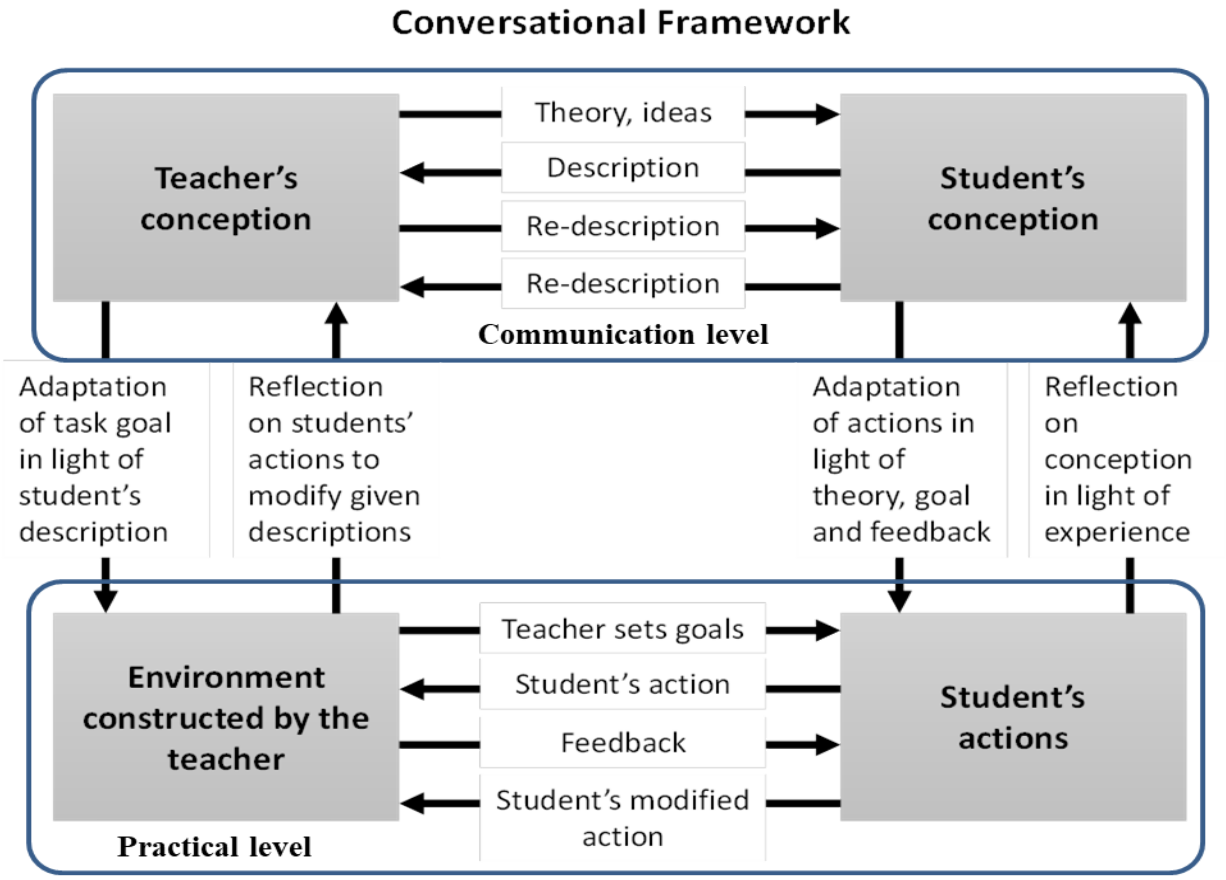

Figure 3.1 The Conversational Framework describe teacher's and student's interaction both on the conceptual level and through exercises offered by the teacher. Both teacher and student adapt their actions based on their communication and experiences through their actions. Teacher creates the environment for the student to practice his knowledge and the student moderates his actions based on the feedback which also affects his conception of the topic.

As a model the Conversational Framework gives a simplified view of teaching, but it has enough detail to cover the most common ways of learning (Laurillard, 2012). It covers learning through acquisition, inquiry, practice, production, discussion, and collaboration. The aim of the framework is to aid the teacher in designing the learning environment. Different kind of technologies can be used in the designs. In this unit we are particularly interested in technology supporting the practice/modelling dimension of the framework (see Figure 3.1 lower half). Educational technology supports the teacher in creating tasks for the pupils that would allow them to practice the learned concepts. The practice dimension relies on the teacher to give feedback on the pupils' performance. Technology can also provide a platform for answers, feedback and grading. In the modelling dimension the teacher provides a technological environment that gives feedback. In the context of disciplinary CT, the late Seymour Papert's work on constructionism (Harel & Papert, 1991) and microworlds (Papert, 1980; 1996) are excellent examples of the modelling aspect.

From a computational perspective the different STEM subjects are answering different questions. Mathematics tells what kind of problems can be solved by computation. Computers were made for calculation and therefore even though the goal is not mathematical the problem have to be expressed with mathematical rigour. Engineering/Technology is about control of electrical devices. For more complex control the device has to have electronics or computational technology embedded. Scientific phenomena can also be modelled as computational processes. All phenomena whose behaviour can be described mathematically can be calculated on the computer. Phenomena that can't be described by exact formulas can be described using statistics. Statistics can handle variation, uncertainty, and large quantities of data. The perspectives answer different questions even though the computational mechanism is the same.

The CT Perspectives framework consisted of pattern recognition, abstraction, decomposition, and algorithms. The chosen technology must be able to represent the pattern that frame the problem. Generally, the pattern represents one of the STEM perspectives. The possibility to use concepts and relationships that are named after the vocabulary of the pattern makes it easier for the pupils to think about a change that would solve the problem. The change from inputs to outputs as a solution to the problem is a description at the highest level of abstraction (problem level). On the object level the pupils define an algorithm that operates based on the concepts and relationships of the pattern. The program level can be closer to the algorithm (object level) or the computer (execution level). The pupils benefit from a programming language or other computational representation that allows decomposition of the problem directly without making transformations between object and program level. The educational technology should provide feedback to the pupils so that they can evaluate the results and make corrections if required.

The Conversational Framework is meant to aid the teacher in developing the learning environment. To disseminate experiences with new learning environments one can, use Pedagogical Patterns (Laurillard, 2012). A Pedagogical Pattern is a way to articulate, test and share the principles and practices of teaching with digital technologies (Laurillard & McAndrew, 2003). The pattern can be used to describe the sequence of teaching-learning activities. Additionally, a Pedagogical Pattern contains information to describe the context of the pattern (see Table 3.1) the origin, summary, topics, learning outcome, rationale, duration, learner characteristics, setting and group size. The Pedagogical pattern is here used as an aid to describe educational technologies for STEAM teaching. The students can also use the patterns later as teachers to share own learning designs with educational technologies.

Table 3.1: Pedagogical Pattern context descriptors (Laurillard, 2012).

Category	**Description**
Origin	the original source and later contributors
Summary	brief description of what is being taught and how
Topics	keywords that will help other teachers decide the relevance to them
Learning outcome	what the learner will know or be able to do by the end
Rationale	the learning approach or pedagogic design principle
Duration	total learning hours, not necessarily continuous
Learner characteristics	educational pre-requisites, experience, interests

Setting	face-to-face, blended, or online
Group size	the range of minimum to any maximum

Examples of educational technologies in STE(A)M

This chapter presents examples of educational technologies for the STEM topics. Each example is presented as a pedagogical pattern which describes a pedagogical design where the technology is a part of a cycle of teaching and learning. The combination of technology and the description of its use form a whole. One can invent a new way of using the technology, but that would also be a new pattern. Additionally, a general description is provided which tells the main characteristics of the pattern and helps the reader to decide if the pattern is worth a closer look. Here follows, examples of patterns for the module topics of mathematics, technology/engineering, and science.

C2STEM (science)

Category	**Description**
Origin	Hutchins, N. M., Biswas, G., Maróti, M., Lédeczi, Á., Grover, S., Wolf, R., ... & McElhaney, K. (2020). C2STEM: A System for Synergistic Learning of Physics and Computational Thinking. Journal of Science Education and Technology, 29(1), 83-100, https://doi.org/10.1007/s10956-019-09804-9.
Summary	The modelling and simulation environment C2STEM is here used to teach physics. The topic is kinematics including the concepts of position, velocity, and acceleration. The models created are computational which supports both learning of physics and CT practices. The modelling environment provide support learning with scaffolded tasks and embedded formative assessments.
Topics	STEM+CT, synergistic learning, Learning-by-modeling, CT, evidence-centered design, open-ended learning environment
Learning outcome	The student will know concepts and practices of physics. The learning by modelling using a computational environment teaches also CT practice like decomposition, refinement and debugging.
Rationale	learning by modelling
Duration	-
Learner characteristics	high-school students
Setting	blended
Group size	1

C2STEM is an open-ended learning environment for STEM learning. It is originated from the evidence centered design principles to design and develop a collaborative C2STEM for integrated learning of STEM + CT in high school science classes (for example Physics in this context). The objective of C2STEM is; learning STEM topics through computational modelling. Computational modelling represents core scientific practices including modelling, verification, and explanation. Computational models assist learning complex systems using mathematics, physics and computer science. C2STEM is based on learning-by-modelling paradigm which supports exploration by enabling learners to stimulate and play with their models and study their behaviour under various conditions. For example, in physics, one example is the use of a computational model to study an atomic structure by exploring its properties under various conditions.

CTSTEM is a web-based and deployed with cloud-based server that allows continuous access through web servers and internet resources. The C2STEM environment is intervened with visual programming (NetsBlox) with domain specific modelling languages to promote synergistic learning of discipline specific (e.g., Physics, Marine Biology) with computer science concepts and practices. Notably, NetsBlox is a visual programming language and cloud-based environment that enables novice learners to create and modify domain specific computational models. NetBlox's visual notation is an extension of Snap and uses JavaScript code to create STEM projects, such as computational models for "kinematics". Figure 3.2 shows a screenshot of a model extracted from (https://www.c2stem.org/).

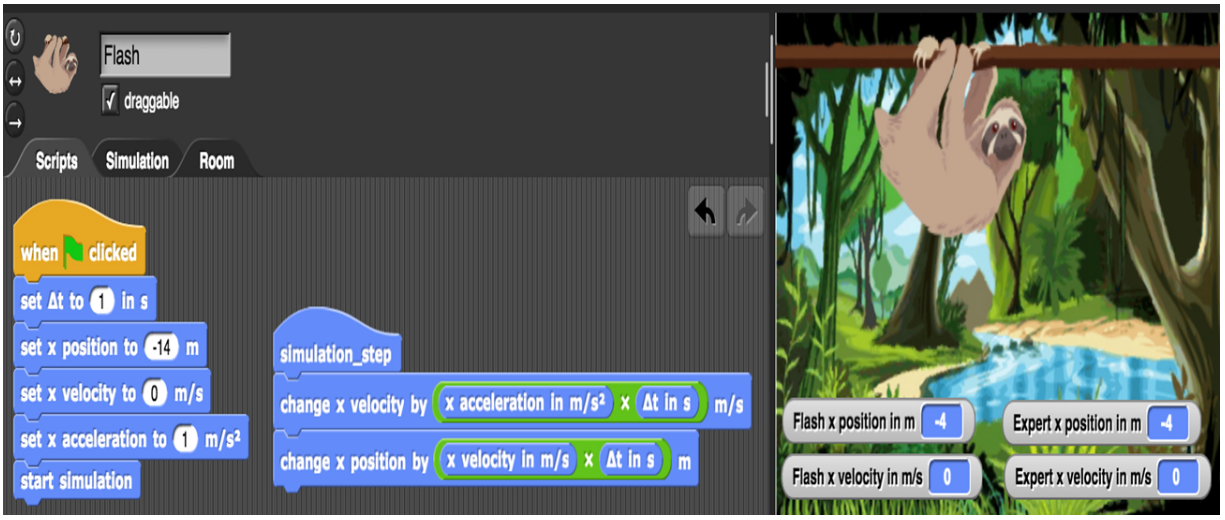

Figure 3.2: Example screenshot for C2STEM based model building

Hutchins et al. (2020) developed a C2STEM system for high schools students to scaffold learning of physics using computational modelling. They adopted a design-based approach to develop and evaluate a collaborative, computational STEM (C2STEM) learning environment. The study framework was set based on the principles: i. Evidence centered design, ii. Learning-by-modelling paradigm in STEM, and iii. Exploratory learning of dynamic processes for CT+ physics learning. Figure 3.3 shows the C2STEM framework developed by Hutchins et al. for their study.

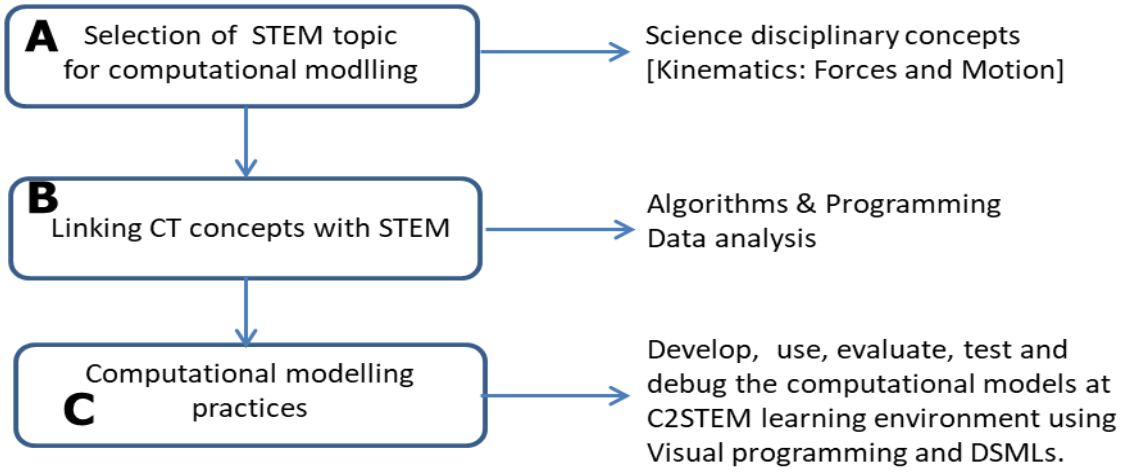

Figure 3.3: Simplified design process developed by Huchins et al.

A. Selection of STEM (Physics in this context) topic for computational modelling:

Topic: Computational modelling of **kinematics**
Sub topic(s) and contents: *Forces and Motion* concepts of position, velocity, and acceleration and their relations including time and distance.

B. Linking CT concepts with a selected topic:

Learning programming to understand basic coding that includes algorithms, initialising and updating variables, operators and expressions, control structures and more. That is, NetsBlox and domain specific programming languages.

C. Computational modelling practices **(A+ B)**

Computational models' theme: **Delivering medicines to a tribe in a remote amazon jungle**
This study adopted the ***problem based learning*** approach to execute the aforementioned process step-by-step to capture student learning outcomes.

The tasks are;

1. Develop a computational model that simulates 1-D, constant **velocity** motion using addition of velocity vectors that occur only under particular conditions.
2. Debug a given model so that it correctly simulates 1-D constant **acceleration** motion by computing velocity and position based on time and acceleration.
3. Use a computational model to solve a 2-D motion problem involving **distance**, **speed**, **acceleration**, and **time**.
4. Use a given graph or data able to *identify possible errors* in a computational model simulating 1-D vertical motion as a result of acceleration due to gravity.

As highlighted (bold and underlined), the first three tasks were designed based on "*kinematics*" topic taught in the Physics classes. Students were instructed to use programming for C2STEM

learning environment to develop, use and evaluate a model to understand the properties of forces and motion under various conditions.

First, students were given the lecture about the theme of computational models and related tasks set for this study. Then, during the model(s) development process students were given *guided inquiry* students were asked to change the parameters of simulation models they developed to evaluate physics relationships. For example, students were asked to document the results of running their models with different positive and negative initial values for acceleration and initial velocity done for task 1. Based on an instructional lecture and guided inquiry, students worked on model building tasks, where they combined their learned physics knowledge with CT concepts to build computational models. In addition, students also applied different conditionals to control the acceleration, speed and direction of the vehicle used in their model. For example, *if the speed of the truck > 50 then "slows down"*. Tasks 3 and 4 were set as challenging for students to extend their exploration skills via learning more coding techniques in align with defined physics concepts set for this study.

SmileyCluster (science)

Category	**Description**
Origin	Wan, X., Zhou, X., Ye, Z., Mortensen, C. K., & Bai, Z. (2020, June). SmileyCluster: supporting accessible machine learning in K-12 scientific discovery. In Proceedings of the Interaction Design and Children Conference (pp. 23-35), https://doi.org/10.1145/3392063.3394440.
Summary	Since the use of Artificial Intelligence (AI) is becoming more common young learners need to understand the nature of the technology to assess its meaning for personal and working life. Machine Learning (ML) is currently one of the major subfields of AI, but due to its advanced mathematical and computational nature it is not accessible for young students. SmileyCluster is an environment where the ML capability of pattern recognition and classification is visualized so that the learner can use his own similar capabilities to understand the functionality of the technology.
Topics	data visualization; hands-on learning; AI literacy; scientific discovery; STEM education
Learning outcome	in general understanding Pattern Recognition (PR) and how ML concepts and methods are related to PR.
Rationale	constructivism, inquiry-based learning, collaborative
Duration	-
Learner characteristics	middle school
Setting	face-to-face
Group size	student pairs

SmileyCluster is web-based collaborative learning environment for learning machine learning (ML) concepts and methods. It uses *k-means* clustering technique for learning basic ML concepts. The *k-means* clustering is one of the most common similarities based unsupervised ML algorithms that widely used for clustering or similarity comparison for data analysis. This clustering method enables data explorations which, in turn, promote hypothesis generation of scientific phenomena. It contains face-overlay, a data visualisation metaphor, which present the translated data points as visual elements for learners to understand the similarity and comparison features of data points.

Wan et al. used *SmileyCluster* to introduce CT integrated data science concepts including ML and *k-means* clustering techniques for high school students. It was conducted as part of pre-college summer program. Twelve students attended the event. This study was facilitated by one course instructor and with four data science researchers. The duration of the program lasted about 2.5 hours. The activities were:

- 25 minutes for class instruction about AI and general differences between supervised and unsupervised learning algorithms
- 15 minutes for pre-study questionnaire (pre-test) about students' background in machine learning and cluster analysis.
- 40 minutes for interaction with the system (*SmileyCluster*): grouping faces, using *k-means* clustering
- 15 minutes for post-study (post-test) questionnaire
- 30 minutes for focus-group (8 students) interview

Refer YouTube video for more details (https://www.youtube.com/watch?v=ZiSR-5zpabw)

The study took place in on-campus computer lab. The web-based interface was setup on desktops in a lab to accommodate all 12 students. As noted, 25 minutes lecture was given to students in the beginning of the session for learning AI and ML algorithms (both supervised and unsupervised methods but clustering). Then, pre-test was conducted focusing on 25-minutes lecture topics and prior knowledge in data science. After a 15-minute pre-study questionnaire students were grouped as pairs of their choices. The groups started using *SmileyCluster* enabled computers. The data set for the study collected from UCI machine learning repository- related to STEM field and validated by scientists. The system already loaded with images (emoji:smiley-icons) as dataset for the exploration.

The objective of the 40-minutes lab session was to 1. Introducing the data set and the face mapping mechanism, and 2. Conveying the concept of multi-dimensional feature space. Course instructor and supporting researchers explained the "Face-overlay" method to students in order to explore, and how to group the face icons based on predefined 16 facial features including eye position, nose texture, brow angles and lips shapes. Face-overlay is the interface of grouping faces is designed for students to explore the clustering process. Students were informed to group the face icons based on listed facial features manually by drag and drop (manual clustering) and to compare the results generated by the system. Notably, Face-overlay supports both pair-wise and global overlay comparisons to see the similarities and differences between data points (face icons in this context) to cluster them. Figure 3.4 show the overview of face-overlay and global-overlay concepts taken from Wan et al. (2020) study.

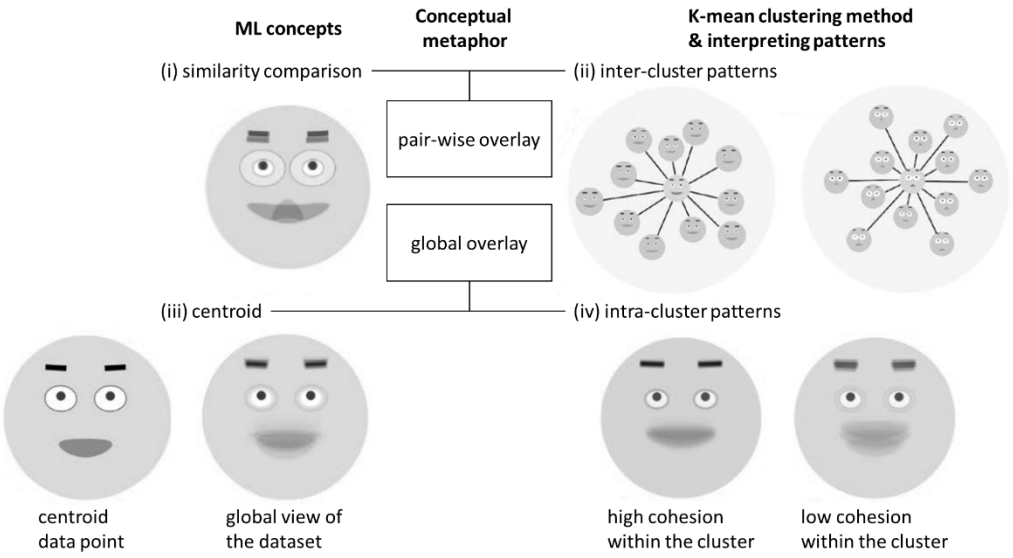

Figure 3.4. The overview of face-overlay metaphor and ML techniques.

In addition to the above, students attended the multiple choice questions to learn about similarity comparison. Figure 3.5 shows the sample of multiple choice questions designed for the study.

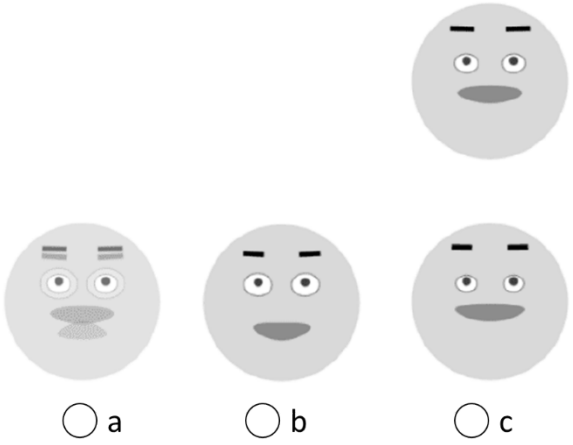

Figure 3.5. The design of multiple choice questions presented at SmileyCluster enabled computer

For example, students were instructed to select the best number for k to learn how *k-means* clustering works based k-inputs (Figure 3.6). That is, comparing the center face and combined the faces of each cluster. Figure 3.6 shows the screen shot of *k-means* clustering based results picture.

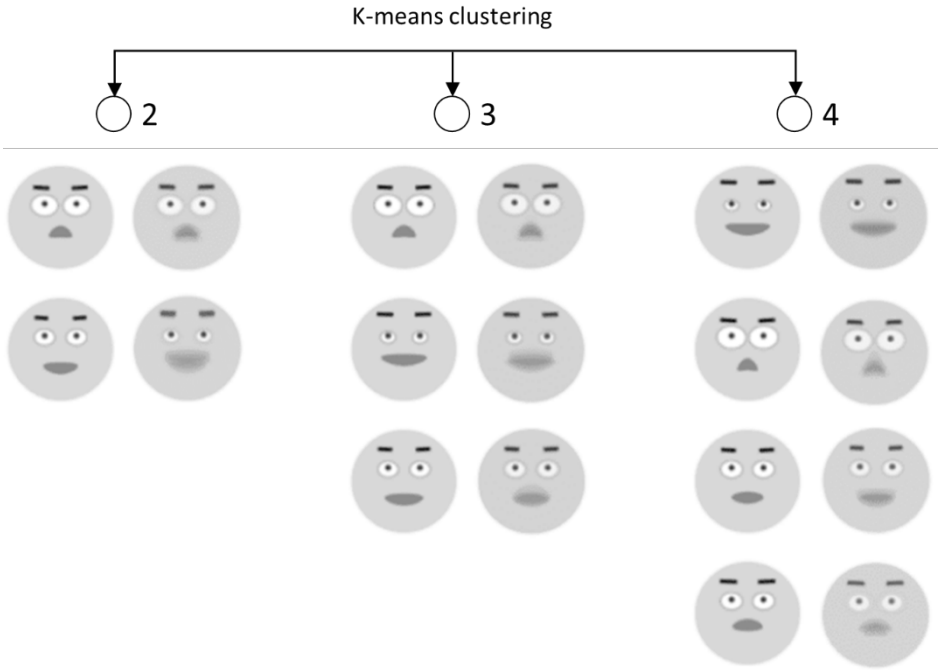

Figure 3.6. The screenshot of k-means clustering defined in SmileyCluster web based environment

After 40-minutes of lab session, post-test was conducted for another 15 minutes to measure student understanding on *k-means* clustering concepts and methods, sense making patterns, and scientific inquiry learned to compare pre and post responses for learning gains. Finally, a semi-structured focus group interview was conducted to obtain participants' view on cluster learning experience in *SmileyCluster* web-based environment. The interview scripts were analysed using thematic analysis. The post-test and interview results suggest that face-overlay design metaphor based *SmileyCluster* environment facilitated understanding the ML concepts, clustering techniques, sense-making pattern recognition and similarly-comparison feature techniques.

Educational Robotics and Creative Computational Problem Solving (subject: technology/engineering)

Category	Description
Origin	Chevalier, M., Giang, C., Piatti, A., & Mondada, F. (2020). Fostering computational thinking through educational robotics: a model for creative computational problem solving. International Journal of STEM Education, 7(1), 1-18, https://doi.org/10.1186/s40594-020-00238-z.
Summary	Educational Robotics (ER) are increasingly used in classroom to teach CT. Creative Computational Problem Solving is a model that supports teachers in the design, implementation, and assessment of ER activities. The model helps the teacher to plan for

	instructional intervention which introduces relevant CT concepts for different phases of ER activities. It gives the student more balanced view of problem solving than just focusing on programming.
Topics	computational thinking, educational robotics, instructional intervention, problem solving, trial-and-error
Learning outcome	The students will know how to control a robot and solve related problems. They will learn how to use the trial-and-error method, problem analysis, idea generation and formulation of solutions in the context of CT.
Rationale	inquiry-based learning
Duration	-
Learner characteristics	primary school students (between 9 and 10 years old)
Setting	face-to-face
Group size	2-3 students

Educational robotics (ER) is a technology used in education to enhance the development of skills and competencies of young learners including children and teenagers. ERs activities are typically contained three main components. They are, 1. ERs, 2. The interface that allow the user to communicate with the robots, and 3. Tasks that to be solved using ERs and interface. Indeed, robotisation is not a new concept in education. ERs are widely implemented in learning settings as part of constructivism [13]. Moreover, integrating robotics in a learning setting can lead to an interest in STEAM topics. ER based teaching and learning to improve learner's critical thinking and problem solving [14]. ERs increasingly used in classrooms for fostering the development of student CT skills. There have been studies discussing how to implement ER activities for CT development in classrooms [15,16]. For example, Chevalier et al. presented a model that guides teachers to identify relevant CT concepts for different phases of ER activities in order to plan suitable instructional interventions. They developed a conceptual framework of educational robotics system (ERS) by combining creative and computational problem solving. It is called creative computational problem solving model (CCPS). It contains five phases. The first three phases of the CCPS focuses on three concepts: understanding the problem, generating ideas, and planning for action (formulating the robot's behaviour). The fourth phase describes the creation of executable code (programming) for the robot and the fifth phase focuses on execution of the code for evaluating the solution (the robot's behaviour).

The proposed CCPS model was evaluated using a robot "*Thymio*" lawnmower (https://www.thymio.org/) with 29 primary school students. The tasks were carried out in the groups of 2-3 students. The task was; execute the lawnmower mission with the *Thymio* robot. Figure 1 shows the playground of the robot lawnmower mission with *Thymio.*

Figure 3.7. (Chevalier et al. 2020; CC BY 4.0)[2] Playground of the robot lawnmower mission with Thymio.

The playground of the robot lawnmower was 45 cm X 45 cm. The fence of the playground was constructed using wood. The lawn area was represented by eight squares of equal size with an imprinted law pattern. The ninth square is imprinted with a brick pattern and placed the bottom right corner of the area, representing a garage-starting point of the *Thymio* lawnmower robot. The robot can be programmed to drive it autonomously around the specified lawn area (Figure 3.7) covering as much as possible. An event-based programming language VPL was used to define the behaviour (programming) of the robot. In VPL platform, the parts of the robot were represented by graphical icons for students to understand and to implement solutions by simple drag-and-drag actions. Figure 3.8 shows the screenshot of VPL programming interface and *Thymio* robot.

[2] Picture has been slightly cropped. Original: Chevalier, M., Giang, C., Piatti, A. et al. Fostering computational thinking through educational robotics: a model for creative computational problem solving. IJ STEM Ed 7, 39 (2020). https://doi.org/10.1186/s40594-020-00238-z. is Open Access licensed by CC BY 4.0.

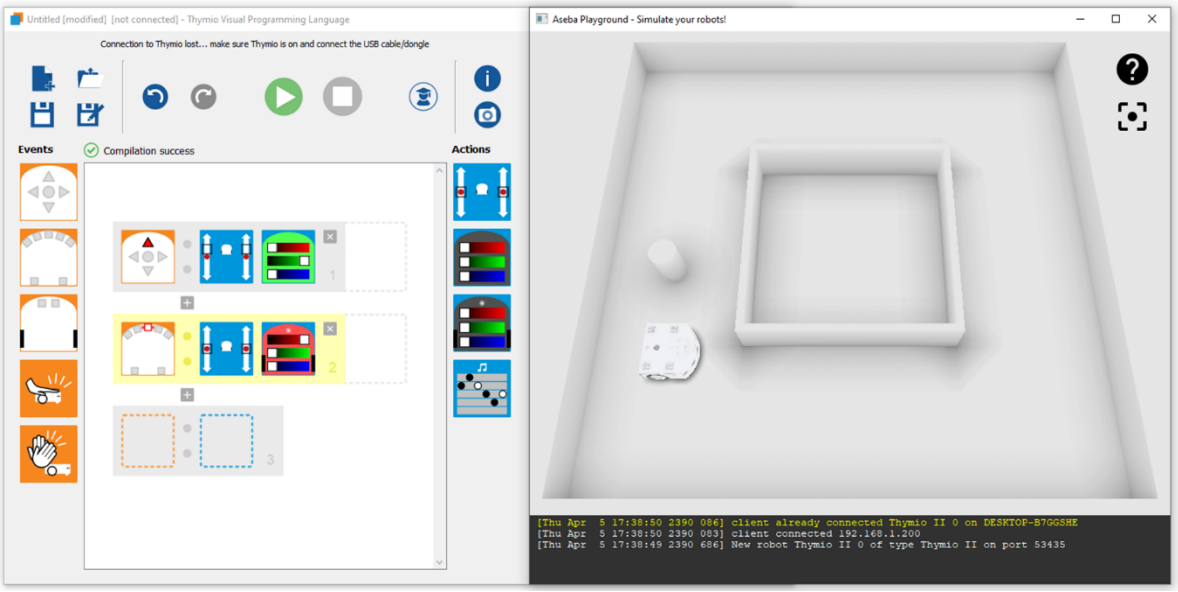

Figure 3.8. Iconic representation of programming commands in VPL platform (left). Thymio simulator (right).

Students were instructed based on CCPS model concepts to proceed one after another. First, all students were introduced to practice with T*hymio* robot and VPL programming interface through several school lessons (1 hour per week for 12 weeks). Students were groped in random as groups of two or three and further classified as test and control groups. The study was conducted in two consecutive sessions of 45 minutes for each experimental condition. That is, the test group started the activities first while control group went for museum exhibition and vice versa. Each group of the students were assigned with one of the two experimental conditions randomly (test or control). The goals and rules of the activities were explained in brief for both groups. The test group started the activity with 10 minutes of blocking of the programming interface. They were given access to the playground and *Thymio* but not allowed to use the VPL programming platform. After this phase, the blocking was released and test groups were allowed to use everything (including VPL programming platform) for next 10 minutes only but not allowed to execute any code on the robot. After 20 minutes of blocking and unblocking the use of VPL programming platform and execution of code, they were allowed to use everything for the last 10 minutes.

On the other hand, control group were given 40 minutes to implement their solutions at lawnmower robot and allowed to experiment as many as times with the T*hymio*, the playground, and VPL programming interface. Both sessions were supervised by two experimenters. They also provided the technical support and addressed students' questions regarding tasks. However, they were not allowed to provide any support on solutions for the task. Both group activities were video recorded for further analysis. The analysis results showed that students spent most of their time on programming and evaluating. They also found that providing unlimited access to programming interface and robots to students promote cognitive processes related to problem understanding, idea generation and solution formulation.

Lattice Land - microworlds for exploring geometry (subject: mathematics)

Category	Description
Origin	Pei, C., Weintrop, D., & Wilensky, U. (2018). Cultivating computational thinking practices and mathematical habits of mind in lattice land. Mathematical Thinking and Learning, 20(1), 75-89, https://doi.org/10.1080/10986065.2018.1403543.
Summary	The subject to be taught is high-school geometry using a mathematical microworld called Lattice Land. The aim is to practice both mathematical habits of mind and CT. The student learns tinkering, experimentation, pattern recognition, and presenting hypothesis in a formal mathematical notation.
Topics	lattice geometry, computational learning environment, mathematical thinking, computational thinking
Learning outcome	Students learn to recognize and reason about complex geometrical patterns. Additionally, they learn computational methods used as part of learning.
Rationale	constructionistic; learning by doing
Duration	-
Learner characteristics	high-school students
Setting	online
Group size	personal

Lattice Land is collection of math microworlds based on NetLogo (Wilensky, 1999) environment. It is intended as a low floor high ceiling introduction to lattice geometry by both offering structure and allowing free exploration. The environment can be used for open-ended experimenting, but it is also suitable for inquiry-based learning if an assignment or question guides the exploration.

The microworlds consist of lattice formed by a matrix of dots. Each dot has a coordinate (x, y) where x and y are integers. One can draw polygons on the lattice by connecting dots with segments. The connected dots form the vertices of the polygon. The learner can interact with the microworld by drawing segments between the dots to form polygons and modifying the polygons by moving its vertices or adding additional segments. Since adding segments and moving vertices can only happen between the dots the lattice provides structure for the exploration. Different microworlds can introduce additional features or constraints. For instance, a microworld can provide information like segment length and polygon area. A constraint would be to allow only manipulation of the vertices of a provided polygon.

The Lattice Land microworlds can be explored on NetLogo web or the environment and the microworlds can be downloaded (https://netlogoweb.org). In the article three microworlds and adjoining questions were described that explored the geometrical concept of area. One example was Lattice Triangles Explore microworld (see figure 3.2) with a four by four lattice and one triangle. The question was how many different sized areas formed by a triangle can be found in this lattice. The student should explore the possibilities by moving the triangle vertices to form different triangles. The correct answer is sixteen, but students can discover also other facts. The students should be encouraged to be mindful of these kinds of findings. One such finding could be that if a single vertex is moved back and forth along the same row or column the area doesn't change. Another one is that if one of the triangles side is parallel to the lattice x- or

y-axis then a triangle with area of 2,5 is impossible to achieve. Students was reported wondering how a triangle with sides that had long decimal tails could still result in nice, rounded area.

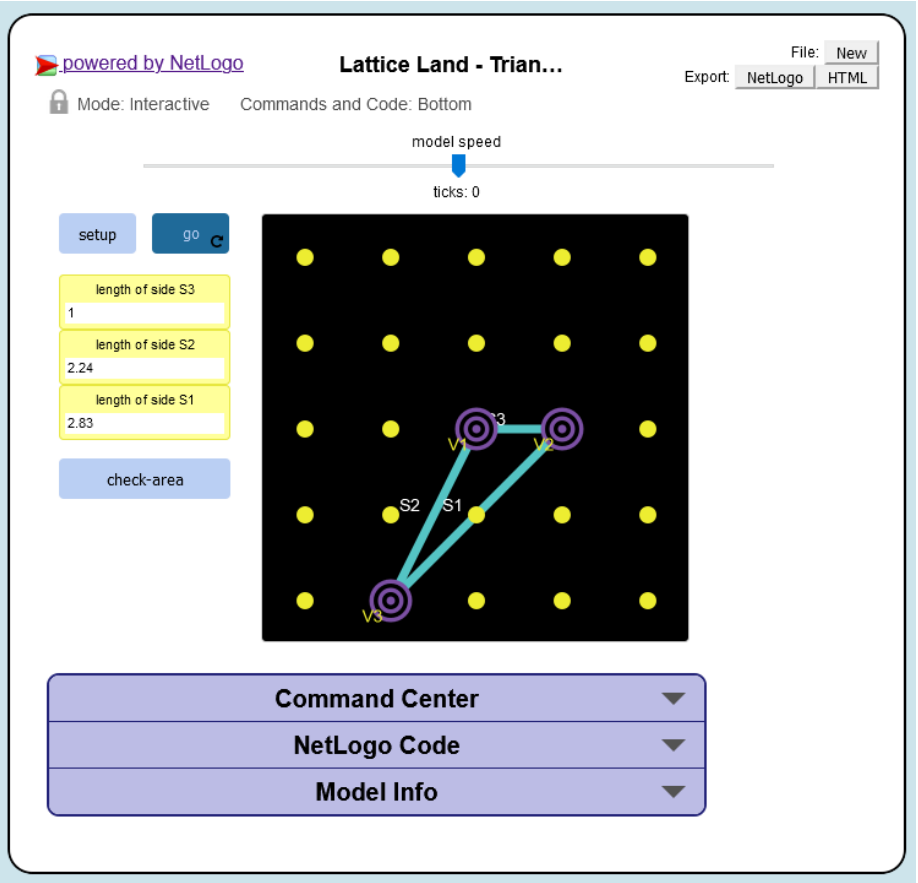

Figure 3.2. Lattice Triangles Explore microworld investigates how many different sized areas formed by a triangle can be found in this lattice.

Lattice Triangles Explore microworld allows the students to explore and develop an understanding of the triangle concept. Especially the relationship between the shape, the length of the sides and the area comes to fore this way. This is the practice of modelling and simulation related to computational thinking (Weintrop et al., 2016). Lattice land utilises interactivity and scaffolding to allow freedom of exploration with meaningful results. This wouldn't be possible or accessible in another medium.

Taxonomy for choosing level of technology use

Based on Module 1/Unit 5: Project Based Learning (PBL) (Valentina Dagiene)

The possibilities of educational technology can be utilised to larger or smaller degree. When starting with a new technology one can first use it as a substitute for an existing one. Later when experience is gained one can exploit the technological capabilities more fully. The Substitution, Augmentation, Modification and Redefinition (SAMR) model (Puentedura, 2006)

provides a four-level taxonomy of selecting, using, and evaluating technology. The model helps thinking about the technologies role in supporting learning.

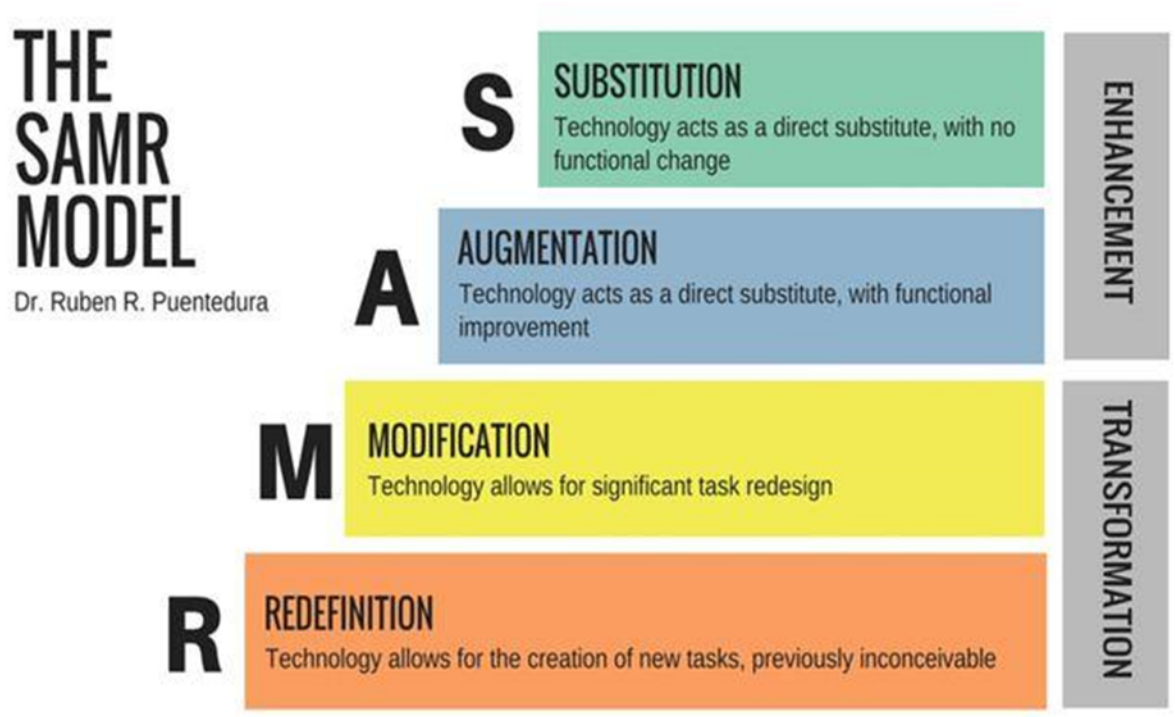

Fig. 5.3. The SAMR model can help educators think about the role of technology in supporting learning (developed by education researcher Ruben Puentedura 2010, Creative Commons)

Substitution
"Substitution" means replacing traditional activities and materials — like in-class lectures or paper worksheets — with digital versions. There is no substantial change to the content, just the way that it is delivered. The goal here is to keep things simple: there's no need to reinvent the wheel. Scan your lessons and worksheets, convert them into PDFs, and post them online using Microsoft OneDrive, Google Drive, or a similar file-sharing service. Think about the information you have on your walls, such as the classroom norms, the daily schedule, or vocabulary lists, and convert them into digital formats that students can easily reference. It may also help to provide synchronous as well as asynchronous versions of your lectures. If you're holding class meetings over a videoconferencing service like Zoom or Skype, provide a recording for students who can't attend. You can also create your own instructional videos for students to view at their own pace.

Augmentation
This level involves incorporating interactive digital enhancements and elements like comments, hyperlinks, or multimedia. The content remains unchanged, but students can now take advantage of digital features to enhance the lesson.

For example, students can create digital portfolios to create multimedia presentations, giving them more options to demonstrate their understanding of a topic. And instead of handing out paper quizzes, you can gamify your quizzes with tools like Socrative and Kahoot.

Teachers can also create virtual bulletin boards—using an app like Padlet—where students can post questions, links, and pictures.

Modification
At this level, teachers can think about using a learning management system like Google Classroom, Moodle, Schoology, or Canvas to handle the logistical aspects of running a classroom, like tracking grades, messaging students, creating a calendar, and posting assignments. Teaching online opens up new channels of communication, many of which can help students who have traditionally been marginalized. Research shows that girls may be less likely to speak up in class, for example, so they may benefit from backchannels—alternative conversations that can run alongside instruction—that encourage participation.

Zoom's text chat feature, meanwhile, gives students an opportunity to write their questions out, which can feel less intrusive if there are dozens of students participating in the call. Also, students who prefer to collect their thoughts may benefit from slower-paced, asynchronous discussions in an online forum or email threads.

Redefinition
Learning is fundamentally transformed at the "redefinition" level, enabling activities that were previously impossible in the classroom, e.g. virtual pen pals can connect students to other parts of the world, whether it's with other students or experts in a field. Virtual field trips enable students to visit locations like the Amazon rainforest, the Louvre, or the Egyptian pyramids. After reading a book in class, you can invite the author to chat about their work and answer questions.

Technology also provides an opportunity to bring authentic audiences into your virtual classroom and can make publishers out of your students. Kids can write their own wikis or blogs for public consumption and feedback—and platforms like Quadblogging can connect distant classrooms together so students both write and respond. Students can tackle local problems—like investigating the water quality of a nearby river—and invite members of the community to assess their digital proposals.

Lecture - Educational technology for CT perspectives

Using the text above
- Applying the CT perspectives
- Educational technologies supporting science
- Educational technologies supporting technology/engineering
- Educational technologies supporting mathematics
- The SAMR model

Activity 3.1 Homework - Planning educational technology support for own topic

1. Select a topic in your subject and plan an exercise utilising "CT perspectives for STEAM" -framework (refer to Unit 2 for details).

2. Choose a suitable educational technology that supports the pupils in performing the exercise (use the educational technologies described in the lecture as a model when you search the literature/Internet).

3. On which level of the SAMR model would you place the use of educationally technology in the exercise (the minimum should be augmented). Revise the exercise design if educational technology doesn't seem to bring any benefit to learning.

4. Fill the table to describe the pedagogical pattern of the exercise you created.

Subject / Topic	
Origin	
Summary	
Topics	
Learning outcome	
Rationale	

Activity 3.2 Homework - Peer reviews

- review Activity 3.1 answers of three colleagues
- Assessment: Accepted/Failed
- Each point is graded on the scale 0-5 (average has to be 2 or better for acceptance)
    - the presentation of the topic
    - argumentation and clarity
    - utilisation of the unit's content

Unit 4 – Creating instructional content for CT integrated STEAM

In this unit, the conceptual model for planning STEAM teaching (Quigley & Herro, 2017) is introduced. The focus is on creating instructional content. The choice of tools should be based on the content, but the tools make it possible to introduce content that would otherwise be too advanced for the pupils. This is also the start of the project work made in teams of 3-4 students, which brings together everything taught in this module. This is the first part of the project where instructional content is created and possibilities for CT application are sought. STEAM

employs a problem-based pedagogy. The problems are open-ended to allow pupils freedom to devise their own solutions utilising the different STE(A)M areas. A in the acronym refers to liberal arts which allows for a broader audience of pupils to engage in problem-solving and helps them to see how the problem relates to the real-world.

Both the instructional content and the learning environment are important in successful STEAM teaching. They are interdependent, the content should make use of the possibilities in the learning environment and in designing the learning environment, the needs of the content should be considered. Here we focus on the content, but prior parts of the module and knowledge of learning environments can be utilised. The purpose is to keep the focus on what knowledge the pupils should gain. In STEAM the problems are taken from the real-world where they often do not have a single right answer. The teacher should create realistic scenarios for presenting the problems to pupils. The possibility for different kinds of answers and the size of the task should inspire creativity and justify the need for collaboration.

Instruction is about how the teacher organizes, prepares, and delivers the content. STEAM content should be designed with three requirements in mind: problem-based delivery, discipline integration and problem-solving skills. Problem-based delivery is central in STEAM pedagogy. Content is presented in relation to the different aspects of the problem. The aspects should be from various disciplines or content areas. The teacher can't be an expert in every discipline, but she can point to resources and other experts which can provide more information. The problem itself shouldn't be a question with one correct answer, but a real-world situation where there are multiple ways to solve the problem. The situation provides the context of the problem and it should feel relevant for the pupils. Authenticity is important, if the problem and its context feel's false the student will be less motivated to learn.

Discipline integration is about teaching the students to combine content and methods from various fields in solving the problem at hand. Even though the idea of STEAM is to use every subject in the acronym, not all problems or problem-solving approaches will require all of them. The choice of content areas and methods should be up to the pupils. The degree of integration of the disciplines in the pupils' answers can vary. It depends on how the problem is presented and the pupils prior experience in combining the topics. The discipline integration can be multidisciplinary, interdisciplinary, or transdisciplinary. Multidisciplinary is the least integrated where the different dimensions of the problem are answered separately. Interdisciplinary problem solving combines the methods from different disciplines. In transdisciplinarity the content of one discipline is made more relevant by looking at it from the contexts of multiple disciplines.

Problem-solving skills describe the characteristics of pupils. There are three type of skills: cognitive, interactional, and creative. By solving problems these skills are developed. The skills are general, but certain types of problems and ways of solving problems can improve a particular skill. Abstracting, analysing, applying, classifying, formulating, interpreting, perceiving, modelling, synthesizing, and questioning are examples of cognitive skills. These can be enhanced by instructional approaches that support observation, experience, reflection, and reasoning. Interactional skills are about communication and collaboration. From a single pupil perspective, the skills include ability to brainstorm, communicate evidence, construct explanations, engage in argumentation, disseminate evidence, present, respond, and explain. To collaborate is to learn together by dividing the tasks related to the problem. Optimally the

collaboration would result in connecting individual knowledge, gathering evidence and joint experience.

Creative learning endeavours are needed to cultivate innovations, ideas, solutions, and productions. The skills needed include designing, patterning, play, performing, modelling, and connecting ideas. STEAM teaching caters for creativity by allowing multiple ways of solving the problems and demonstrating understanding. The teacher must offer the pupils concepts and tools to engage in the problem-solving scenarios and gain experiences. Both the Art and Technology subjects in the STEAM acronym allows creativity. By combining these with the other topics, new forms of expression can be found which supports both skill development and content understanding. Design is the combination of art and technology in creating products as solutions to problems. Sometimes usability of a technology is a problem which design can solve.

CT touches on many of the aspects described in STEAM pedagogy. It incorporates the views of the different subjects and provide tools to explore them. Even though controlling a computer is not artistic (there can be differing opinions) it can be used as an artistic medium. Computing can be used to simulate different phenomena or technologies if they are not available. Different data sets and results can also be visualized. Computer presentation are not just text and pictures, but also animations, sounds and videos. To choose and combine the many capabilities of computers requires creativity. Like, the different subjects of STEAM, the possibilities of CT and related tools should not be forced on the pupils but be available when needed. The pupils need prior experience with CT and computing tools to be able to utilise them in STEAM projects.

Example STEAM exercise 1

“It is estimated that only 1 in 1,000 sea turtle hatchlings will survive to adulthood (http://www.seeturtles.org/baby-turtles/). Our local Sea Turtle Hospital is one place that helps to rescue, rehabilitate and release sea turtles back into the ocean. Why do you think so few sea turtles reach adulthood? What are some problems they face? Could people be unintentionally harming sea turtles? What can we do to help? You and your team will work together to decide how you want to educate our school. You will research what is hurting sea turtles, as well as what may be killing them. You may want to contact the Sea Turtle Hospital, and see if they can answer some of your questions, as well as tell you what hurt the sea turtles they are rehabilitating. Consider creating a graph of your findings to see if there are any trends. Design a two-part presentation; the first part should present your findings and the second part should be a plan of action for how we can help the sea turtles. You could use an app like ChatterPix Kids or Toontastic to record interviews or as part of your presentation. You will decide how to present your findings to the class. We will then determine as a class which plan to implement.” (Quiqley et al. 2020, Electronic supplementary material 2)

Example STEAM exercise 2

“The next generation of human device interfaces (i.e. eye tracking; face recognition, iWatch) is embedding technology into clothing, or e-textiles. Such a step could bring people and machines closer together. Such devices could also change the design of clothing, such as

inbuilding cameras. Elite athletes are interested in these because of their ability to detect muscle fatigue which has implications for training. The healthcare industry is hoping this will help patients with disease such as epilepsy, diabetes, asthma, and more. However, the e-textile market would like to break into one of the largest un-tapped markets...teens! They have contacted PSM to hear your pitches (ideas) for the use of e-textiles. With your team, you will research an area where e-textiles could be used in your life-- your pitch needs to include a prototype, use, and discussion of some of the challenges of e-textiles and your plan for remedying those issues." (Quiqley et al. 2020, Electronic supplementary material 2)

Example STEAM exercise 3

"As more people move to XXXX and businesses continue to grow, traffic is increasing and causing delays, especially along XXXX Road. For example, this means that a person driving 10 miles from their home on XXXX to XXXX High School might spend 45 minutes in traffic for what should take less than 20 minutes. This comes at a great cost to families who spend less time together, the environment which is polluted by increased and slower traffic, and the economy as work time is reduced.

As a city planner, you have been given the challenge of designing a solution to the traffic problems in the local area. You will use your knowledge of the scientific method and engineering design process to find a solution and test it. You have decided it will be helpful to think about the history of XX and how it has changed since colonial times to get a better idea of the land and transportation needed. Also, you will research and read articles on the geographic area, traffic, and the increased houses and businesses expected for XXXX. You will need to propose a plan to alleviate traffic issues along with a model to show the city council, but you will also need to be prepared to share the information with the community, as well.

Driving question: How can studying history and traffic patterns in a region help to design a solution to transportation problems?" (Quiqley et al. 2020, Electronic supplementary material 2)

Lecture - STEAM Teaching Model 1/2

Based on the material (text above) and Unit 2: CT Perspectives for STEAM framework

- Problem-based delivery
- Discipline integration
- Problem-solving skills
- Application of CT Perspectives
- Case studies

 The conceptual model of STEAM teaching – Instructional content

Quigley, C. F., Herro, D., & Jamil, F. M. (2017). Developing a conceptual model of STEAM teaching practices. School Science and Mathematics, 117(1-2), 1-12. https://doi.org/10.1111/ssm.12201

Activity 4.1 course project part 1/2

This the beginning of the course project. The goal is to plan a learning intervention. The project is divided in two parts. In this part you will assemble the project team and plan the content of what you are going to teach. The size of the project can vary depending on the instruction given by your teacher.

You should make a team with 3-4 of your colleagues. It is beneficial if your team would have people representing different subjects or alternatively some of the members could represent the viewpoint of a subject other than their own. Choose a topic which is timely, combines different subjects, has a real-world context, and is relevant to expected pupils (choose a target group). Create the task description and plan what kind of information the pupils could need. Remember that the task should give space for creativity and alternative solutions. Also keep CT in mind, the topic should have possibilities to utilise computational approaches.

Summary

- in teams of 3-4 students
- topic selection
- task description
- creating instructional content
- recognizing opportunities for applying CT

Activity 4.2 team discusses the plan with the supervisor

- When the plan is ready.
- The team presents the plan to the supervisor.

Unit 5 – Designing the learning environment for CT integrated STEAM

This unit continues with the conceptual model for planning STEAM teaching (Quigley & Herro, 2017; Quigley & Herro, 2019). The focus is on designing the learning environment. The

other units in this module have additionally given the bases to choose educational technology that supports the integration of CT to STEAM topics. Like in content planning, the focus shouldn't be on the technology, but on creating an environment where the scenario can be enacted. Since the approach of problem solving is up to the pupils then likewise the technology used should also be their decision. By incorporating problems where automating algorithms would be a natural solution the pupils are encouraged to think computationally. This unit is also the start of the second part of the project work where the learning environment is designed, and educational technologies are chosen for the scenario(s) created in the previous unit. A design for a STEAM learning environment should focus on instructional approaches, technology integration, assessment practices, and equitable participation.

STEAM teaching is based on teacher-facilitated learning, which differs from the traditional teacher-led lessons. Pupils should be encouraged to take charge and responsibility for their own learning. This can be difficult in the beginning and pupils can still look to the teacher for answers. One method that can be used to change this behaviour is that the pupils should ask or lookup three sources before contacting the teacher. The students should learn to rely on the knowledge of their peers or on what they themselves can find out. In addition, before the teacher answers the pupils, they could be asked to tell what they have found so far. The teacher shouldn't provide a correct answer but build on the reported findings and encourage to explore further. Real-world problems don't have a definite answer and pupils should learn to be comfortable with not knowing an exact one. However, the answers can be more or less well prepared and here the pupils' ability to justify the chosen solution is important.

Lessons develop skills such as abstracting, analysing, applying, formulating, collaborating, engaging in argumentation, disseminating evidence, and presenting. The teacher should regularly create opportunities for the pupils to use these skills in different contexts. According to the STEAM teaching model the pupils should learn to explore multiple paths in solving a problem. Freedom to choose the problem-solving approach allows for inventiveness and use of creative skills. To support this the teacher should augment the problem-solving scenario with concepts, tools and opportunities for various experiences. The problem description should support student-guided learning where peer assistance and collaboration comes naturally. This requires that the problem design allow pupils to assume different roles in solving the problem. When problem-solving requires combinations of different knowledge and skills then asking help from peers and collaboration doesn't feel forced. The teacher should design the problem scenario to encourage age-appropriate levels of social and emotional engagement in learning.

Technology is one of the subjects of STEAM. In creating the learning environment, the pupils should have access to different kinds of technological tools for solving the problem. They shouldn't be users but producers of technical solutions. To apply technology to the problem the pupils need prior experiences with the tools in relation to the issues in the problem scenario. Only by being comfortable with the technology, it can be integrated in the learning. In the context of this module, the emphasis is especially on tools that support CT integration. The solution has to have aspects that benefit from automating the execution of an algorithm. One must differentiate between usage of readymade solutions and creating the solution. As an example, presenting historical facts on a web page is not CT, but discovering those facts through analysing data computationally is. The pupils can benefit of having tools for particular purposes and general tools that can be modified (programmed) to fit the need. The general tools

can be more supportive of combinations of methods from different subjects since they are not made for single context.

Assessment is a central part of any educational model and therefore also in STEAM teaching. The instruction, learning and assessment must be aligned to meet the goals of education. Student-driven aspect of STEAM learning makes traditional testing methods like multiple-choice tests misaligned. The choice of method should reflect the STEAM model where the goal is for the pupils to engage in multiple modes of inquiry, learn to use a wide variety of skills, and collaborate in finding the solution. To be authentic the pupils should be asked to apply the knowledge and skills they learned in the context of the problem they are solving. The problem-solving process that ends in a solution is indication of gained knowledge. However, to just assess the result leaves out the possibility to improve the skills used while the process is ongoing. To ensure that the assessment is linked to STEAM teaching practices it should be embedded in the learning process. The teacher should give frequent and high-quality feedback to the pupils during problem-solving. In this way, the teacher can ensure that the pupils understanding of content is aligned with the learning objectives. The feedback can also encourage pupils to be more mindful of their progress and think more deeply on the various topics.

The goal of equitable participation is to be fair and reasonable in relation to the pupil's prior skills and knowledge. Cultural norms and family traditions affect the background knowledge the pupils possess. These should be taken into account so that everybody starts from a common ground. By providing mentors and experts, the teacher can support the pupils that might not have the opportunity to access that kind of knowledge outside school. It can open the student's eyes for different kind of views and opportunities for future occupations. Here also a cultural sensitivity is important so not to promote stereotypic views but show that a future can be what you make of it regardless of your background. By creating space for self-expression, the teacher allows the student to display their strengths. Since STEAM teaching is student-directed and the assessment is embedded in the learning process this allows for a more equitable participation by not forcing every pupil to the same assessment model.

Lecture - STEAM Teaching Model 2/2

Based on the material (text above) and Unit 3: Choosing educational technologies based on CT Perspectives

- Instructional approaches
- Assessment practices
- Equitable participation
- Choosing educational technology for CT

 The conceptual model of STEAM teaching – Learning environment

Quigley, C. F., Herro, D., & Jamil, F. M. (2017). Developing a conceptual model of STEAM teaching practices. School Science and Mathematics, 117(1-2), 1-12. https://doi.org/10.1111/ssm.12201

Quigley, C. F., Herro, D., Shekell, C., Cian, H., & Jacques, L. (2020). Connected learning in STEAM classrooms: Opportunities for engaging youth in science and math classrooms. International Journal of Science and Mathematics Education, 18(8), 1441-1463. https://doi.org/10.1007/s10763-019-10034-z

Activity 5.1 Course project 2/2

Here the project started in Unit 4 continues. When you have the exercise content and task ready you should start to plan the implementation. The pupils need information to complete the exercise and there are different ways to provide this. You can plan lectures, auxiliary material and suggest internet sites or keywords to use in the search for information. Access to experts would also strengthen the authenticity of the exercise. Another part of the implementation is the technological support needed. There are subject related equipment and the usual office software etc. Since we are here looking at integrating CT a special attention should be given to make available computational tools fit for the intended purpose.

Summary

- Continues from unit 4.
- Designing the learning environment
- Planning educational technology support
    - especially support for CT
- Writing a report
- Preparation of presentation

Unit 6 – Project presentations

Presentation of the prospective teacher teams learning intervention designs (project work).

Activity 6.1 Presentations and discussion

Own team presentation 15 min

time allocated for each presentation.


10 min presenting design.


5 min questions and comments by the audience.

Assessment requirements and assessment strategy

All assessment tasks should be handed in before the deadline set.

Assessment task should measure and provide evidence about the achievement of learning outcomes of the module	**Assessment criteria and method** for written assignments: e.g. lengths (in words), structure (introduction, main part, conclusions), proper use of terms and concepts.
Review of plan (based on PRADA) for applying CT to STEAM topic(s)	Accepted/Failed, peer review. Criteria: the presentation of the topic, argumentation and clarity, utilisation of the unit's content.
Review of plan (based on CT Perspectives for STEAM framework) for how to choose right technology supporting learning of STEAM topic(s)	Accepted/Failed, peer review. Criteria: the presentation of the topic, argumentation and clarity, utilisation of the unit's content.
Instructional design project report and presentation	The lecturer gives the course grade based on the project work. The project report is assessed based on the "STEAM Classroom assessment for Student Learning Experience" -criteria (see below). How well the project presentation summarises the main ideas can give an optional + to the final grade (in the Finnish grading system scale 0-5 this would be 0,25).

Peer reviews

The homework in Unit 2 & 3 are peer reviewed with the grades Accepted/Failed. Both need to be accepted.

- Each point is graded on the scale 0-5 (average has to be 2 or better for acceptance)
  - the presentation of the topic
  - argumentation and clarity
  - utilisation of the unit's content

STEAM Classroom assessment for Student Learning Experience (modified with criteria for CT)

An explicit assessment criterion is used to grade the instructional design project created in Units 4 & 5 (0-5). The presentation in of results of the project Unit 6 can give an additional + (0,25).

Read the document *TeaEdu4CT_8_module_assessment.xlsx* (for the original see the reference below Assessment is based on…)

Grading of the project work (0-5 each criterion):

- Subject-Matter Alignment
- Discipline Integration
- Problem-Solving Skills
- Computational Thinking
- Instructional Approaches
- Assessment
- Equitable Participation

Assessment is based on:

Quigley, C. F., Herro, D., Shekell, C., Cian, H., & Jacques, L. (2020). Connected learning in STEAM classrooms: Opportunities for engaging youth in science and math classrooms. International Journal of Science and Mathematics Education, 18(8), 1441-1463. https://doi.org/10.1007/s10763-019-10034-z

ESM 1 (Electronic supplementary material; https://doi.org/10.1007/s10763-019-10034-z)

How well the project presentation (Unit 6) summarises the main ideas can give an optional + to the final grade (in the Finnish grading system scale 0-5 this would be 0,25).

Implementation ideas

Unit 4 and 5 makes it easy to extend or shorten the module by making the project work more elaborate or simple. One can extend the project work by designing a set of learning interventions or a whole course. It is also easy shorten the module by offering the students readymade materials and templates. In the extreme case the project work could be an individual assignment ideating a learning intervention and presentation of the ideas in Unit 6.

References

Bermúdez, J. (2020). Cognitive Science: An Introduction to the Science of the Mind (3rd ed.). Cambridge: Cambridge University Press.

Blockley, D. (2012). Engineering: a very short introduction. OUP Oxford.

Checkland, P., & Holwell, S. (1998). Information, systems, and information systems. Chichester: John Wiley & Sons.

Chevalier, M., Giang, C., Piatti, A., & Mondada, F. (2020). Fostering computational thinking through educational robotics: a model for creative computational problem solving. International Journal of STEM Education, 7(1), 1-18.

Dasgupta, S. (2016). Computer science: a very short introduction (Vol. 466). Oxford University Press.

Davis, M., Sigal, R., & Weyuker, E. J. (2015). Computability, complexity, and languages: fundamentals of theoretical computer science. 2. edition. Elsevier.

Denning, P. J. (2007). Computing is a natural science. Communications of the ACM, 50(7), 13-18.

Denning, P. J. (2017a). Computational Thinking in Science. American Scientist, 105(1), 13-17.

Denning, P. J. (2017b). Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6), 33-39

Devlin, K. (1994). *Mathematics: The science of patterns: The search for order in life, mind and the universe*. Macmillan.

Devlin, K. J. (2012). *Introduction to mathematical thinking*. Palo Alto, CA: Keith Devlin.

Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., ... & Andrews, A. (2019). PRADA: A Practical Model for Integrating Computational Thinking in K-12 Education. *In Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 906-912).

Gilbert, J. and Boulter, C. (1998). Learning Science Through Models and Modelling. International Handbook of Science Education vol 1 ed B J Fraser and K G Tobin (Dordrecht: Kluwer Academic) pp 53–66

Gowers, T. (2002). Mathematics: A very short introduction (Vol. 66). Oxford Paperbacks.

Gutiérrez, R. and Pintó, R. (2005). Teachers' conceptions of scientific model. Results from a preliminary study ESERA 2005 Conf. Proc (Noordwijkerhout, Netherlands)

Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of computer programming, 8(3), 231-274.

Harel, I., & Papert, S. (Eds.). (1991). Constructionism. Ablex Publishing.

Hersh, R. (2014). Experiencing Mathematics: What do we do, when we do mathematics? (Vol. 83). American Mathematical Soc.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS quarterly, 75-105.

Hill, R. K. (2016). What an algorithm is. Philosophy & Technology, 29(1), 35-59.

Hutchins, N. M., Biswas, G., Maróti, M., Lédeczi, Á., Grover, S., Wolf, R., ... & McElhaney, K. (2020). C2STEM: A System for Synergistic Learning of Physics and Computational Thinking. Journal of Science Education and Technology, 29(1), 83-100.

IEEE (2010) Iso/iec/ieee 24765:2010 systems and software engineering - vocabulary

Jocius, R., Joshi, D., Dong, Y., Robinson, R., Cateté, V., Barnes, T., ... & Lytle, N. (2020). Code, Connect, Create: The 3C Professional Development Model to Support Computational Thinking Infusion. *In Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 971-977).

Justi, R. S. and Gilbert, J. (2002). Modelling, teachers' views of the nature of modelling, and implications for the education of modellers Int. J. Sci. Educ. 24 369–87

Kramer, J. (2007). Is abstraction the key to computing? Communications of the ACM, 50(4), 36-42.

Kvasz, L. (2019). How Can Abstract Objects of Mathematics Be Known?. *Philosophia Mathematica*, 27(3), 316-334.

Laurillard, D. (2002). Rethinking University Teaching in the Digital Age.

Laurillard, D. (2012). Teaching as a design science: Building pedagogical patterns for learning and technology. Routledge.

Laurillard, D., & McAndrew, P. (2003). Reusable educational software: A basis for generic learning activities. Reusing online resources: A sustainable approach to e-learning, 81-93.

Lopez, V., & Hernandez, M. I. (2015). Scratch as a computational modelling tool for teaching physics. Physics Education, 50(3), 310.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. ACM Transactions on Computing Education (TOCE), 10(4), 1-15.

Malyn-Smith, J., Lee, I. A., Martin, F., Grover, S., Evans, M. A., & Pillai, S. (2018). Developing a framework for computational thinking from a disciplinary perspective. In Proceedings of the International Conference on Computational Thinking Education (p. 5).

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. AI magazine, 27(4), 12-12.

Miller, G. A. (2003). The cognitive revolution: a historical perspective. Trends in cognitive sciences, 7(3), 141-144.

NRC (National Research Council). 2012. A framework for K–12 science education: Practices, crosscutting concepts, and core ideas. Washington, DC: National Academies Press.

NGSS Lead States. 2013. Next Generation Science Standards: For states, by states. Washington, DC: National Academies Press. www.nextgenscience.org/next-generation-science-standards.

Okasha, S. (2016). Philosophy of Science: Very Short Introduction. Oxford University Press.

Papert, S. (1980). Mindstorms: Computers, children, and powerful ideas. NY: Basic Books, 255.

Papert, S. (1996). An exploration in the space of mathematics educations. International Journal of Computers for Mathematical Learning, 1(1), 95-123.

Pears, A. (2019). Developing Computational Thinking," Fad" or" Fundamental"?. *Constructivist Foundations*, 14(3).

Pei, C., Weintrop, D., & Wilensky, U. (2018). Cultivating computational thinking practices and mathematical habits of mind in lattice land. Mathematical Thinking and Learning, 20(1), 75-89.

Perrenet, J., Groote, J. F., & Kaasenbrood, E. (2005). Exploring students' understanding of the concept of algorithm: levels of abstraction. ACM SIGCSE Bulletin, 37(3), 64-68.

Puentedura, R. (2006). Transformation, technology, and education [Blog post]. Retrieved from http://hippasus.com/resources/tte/.

Quigley, C. F., Herro, D., & Jamil, F. M. (2017). Developing a conceptual model of STEAM teaching practices. *School Science and Mathematics*, 117(1-2), 1-12.

Quigley, C. F., & Herro, D. (2019). An Educator's Guide to STEAM: Engaging Students Using Real-World Problems. Teachers College Press.

Quigley, C. F., Herro, D., Shekell, C., Cian, H., & Jacques, L. (2020). Connected learning in STEAM classrooms: Opportunities for engaging youth in science and math classrooms.

International Journal of Science and Mathematics Education, 18(8), 1441-1463. https://doi.org/10.1007/s10763-019-10034-z

Rich, P. J., Egan, G., & Ellsworth, J. (2019, July). A Framework for Decomposition in Computational Thinking. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (pp. 416-421).

Sfard, A. (2007). When the rules of discourse change, but nobody tells you: Making sense of mathematics learning from a commognitive standpoint. The journal of the learning sciences, 16(4), 565-613.

Tedre, M. (2018). The nature of computing as a discipline. *Computer science education: Perspectives on teaching and learning in school*, 2.

Tedre, M., & Moisseinen, N. (2014). Experiments in computing: A survey. The Scientific World Journal, 2014.

Thimbleby, H. (2007). Press on: Principles of Interaction Programming.

Wan, X., Zhou, X., Ye, Z., Mortensen, C. K., & Bai, Z. (2020, June). SmileyCluster: supporting accessible machine learning in K-12 scientific discovery. In Proceedings of the Interaction Design and Children Conference (pp. 23-35)

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. Journal of Science Education and Technology, 25(1), 127-147.

Weintrop, D. (2019). Block-based programming in computer science education. Communications of the ACM, 62(8), 22-25.

Wild, C. J., & Pfannkuch, M. (1999). Statistical thinking in empirical enquiry. International statistical review, 67(3), 223-248.

Wilson, K. G. (1989). Grand challenges to computational science. Future Generation Computer Systems, 5(2-3), 171-189.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

Wolff, A., Gooch, D., Montaner, J. J. C., Rashid, U., & Kortuem, G. (2016). Creating an understanding of data literacy for a data-driven society. The Journal of Community Informatics, 12(3).