Computational Thinking

A New Approach for Teaching Computer Science to College Freshmen

Linnea Wärmedal

Linnea Wärmedal
VT 2014
Bachelor thesis, 15 credits
Supervisor: Jerry Eriksson
External Supervisor: Johanna Björklund
Examiner: Pedher Johansson
Kandidatprogrammet i datavetenskap, 180 hp

Abstract

Felix, qui potuit rerum cognoscere causas

The traditional way of introducing computer science to college freshmen is through a programming course. Such a course often account for programming, problem solving, efficiency, debugging, memory allocation and complexity. The student is presented to all of this within the first course in computer science. To be introduced to all these concepts during the first course could be compared to learning fundamental arithmetic alongside the mean value theorem. What if the student, instead of learning the traditional, where to learn the basics of computer science? What if programming didn't come first?

If the student where to get proficient in the basics of computer science they would be able to focus on the specifics of the different field of computer science.

In this thesis I present a proposition of a syllabus for a college freshmen course to be taught to computer science minors. The course will be built on the concepts of computational thinking.

The concept of computational thinking and how it is used in education will also be account for.

Keywords Computational Thinking, Computational Thinking in Education, Computer Science Education

Komputationellt Tänkande - Ett nytt sätt att introducera datavetenskap

Sammanfattning

Det traditionella sättet att introduceras till datavetenskap är genom en programmeringskurs. Denna kurs innehåller programmering, problemlösning, effektivitet, felsökning, minnesallokering och komplexitet. Allt detta ska studenten lära sig, eller i alla fall gå igenom, under sin första kurs i datavetenskap. Att introduceras till alla dessa koncept under en introducerande kurs skulle kunna liknas med att lära sig fundamental aritmetik samtidigt som medelvärdessatsen. Om studenten istället får gedigna baskunskaper skulle denne kunna fokusera på mer specifika kunskaper inom de olika områdena.

Som en lösning på detta problem presenteras i denna rapport ett förslag på en första kurs inom kandidatprogrammet i datavetenskap som bygger på konceptet komputationallt tänkande och dess användning.

En redogörelse av konceptet komputationellt tänkande och hur det används i utbildning idag kommer även att presenteras.

Keywords Komputationellt Tänkande, Komputationellt Tänkande i Utbildning, Undervisning i Datavetenskap

Acknowledgements

There is a time in everyone's life when life brings you down. Without the support from friends and family this thesis would never have been written.

To my precious family for the unconditional love and support you gave me when I reached out for the stars! To Björn for always having my back.

When given a space, support and curious questions one can truly rise to the challenge for answers. Given that, combined with a solid rock and interested feedback one can truly fly! That's why I want to send a special thanks to Codemill, for beliving in me and for the support and interest in my thesis. And for the support you gave me when life struck hard during the course of my thesis. A special thanks to Johanna Björklund for the feedback and scientific guidance you gave throughout my work. And to Erik Långström, for adding a spark to my work!

And for Håkan, Susanne and Daniel for teaching me how to use my words, to precious knowledge and for your never-ending belief in me when I needed it the most.

Finally I want to thank Jerry Eriksson for giving me the article that forever changed my belief in myself as a computer scientist; without which this seed never would have been planted.

Contents

List of Figures

List of Tables

1 Introduction

When studying a new field one usually starts with the basics. In math one start with counting and number recognition, when learning a foreign language one starts with words. Only after studying the basics does one enter higher education in the field. As the old cliché goes "Practice makes perfect" stating that in order to achieve perfection one must practice. In higher education this becomes crucial for good results. But what is basic knowledge in the field of computer science? Wikipedia defines computer science as follows[1]:

> *Computer science (also called computing science) is the study of the theoretical foundations of information and computation and their implementation and application in computer systems.*

Figure 1: The overwhelming feeling when combining programming and computational process

This outline holds a lot of knowledge and what the first step towards computer science is differs. When entering the bachelor program in computer science at Umeå University the first course in computer science is "programming in the C language" [2]. A first study of the curriculum of the course shows that the student is expected to learn the following:

- Translate simple algorithms to the given programming language
- Understand and use variables, expressions and constructions in a high level language.
- Construct and use functions to write a well structured C program

- Use elementary data-types, arrays and strings and to know their limitations
- Understand the principles of and perform error diagnosis.
- Use dynamic memory allocation and file management

This course does not only focus on the student's ability to translate instructions into code, it also focuses on the student's ability to understand the language, decomposition, algorithms and to make decisions about the design of the program.

In 2009 James J. Lu and George H.L Fletcher wrote[3]:

> *Programming is to CS what proof construction is to mathematics and what literary analysis is to English.*

With this quote Lu and Fletcher were trying to show that programming is not basic knowledge in computer science but a tool to develop higher knowledge. Instead they talked about another kind of knowledge as fundamental - computational thinking, which led to the following conclusion:

> *Substantial preparation in computational thinking is required before students enroll in programming courses.*

Jeanette Wing, a pioneer in the field of computational thinking, stated the importance of basic knowledge of the characteristics of computational thinking at an early age [4]. Computational thinking knowledge would allow the student to systematically and efficiently process information and tasks a skill necessary for the people in the twenty-first century [3].

In computer science computational thinking would be the logic behind the science, applicable in all fields of computer science. If we have such a cross field knowledge, why aren't we teaching it to our freshmen?

In USA the call for a new skill has led to the process of integrating computational thinking into the K-12(Kindergarten to the 12th year of school) curriculum. The aim of the process is to teach students this skill at an early age, giving them the tools for life in the digital world. [5]

1.1 Relation to gender aspects in Computer Science

In the years 2012/2013, 940 students at Swedish universities were awarded a bachelor of science in informatics, computing science or system analysis. Only 244 of which are women [6]. When entering an education in computer science one can truly point out the absence of women in the field. In everyday life however, via computers at work, social networks and surfing women are highly represented [7]. The strive to get more women into computer science has necessitated a work of changing the attitude towards computer science, what and for whom it is.

Eric Roberts et al. [8] reported the work at Stanford university to encourage more women to major in computer science. One of the strategies was to develop an introduction course to appeal to a broader audience. The many reports [9, 10, 11, 12]

of the broadness of computational thinking as a cross field subject would make computational thinking a fair candidate for such a course.

There is reason to believe that bringing computational thinking into daylight might change the attitude towards and enrich the broadness of the field and therefore attract more women to major in computer science. In USA computational thinking is integrated into the K-12 curriculum and the hopes are that children whom at an early age are given access to a computer and taught computing skills will later grow an interest in computer science. This is beneficial for women since statistics show that access to resources are less given to women at an early age. [7]

1.2 Statement of the Problem

The recognition for computational thinking as a skill for everyone [4] has led to the work of bringing computational thinking into the K-12 curriculum in the USA [9]. In Sweden computational thinking is not yet taught in neither elementary school nor upper secondary school. In their report J. Lu and G. Fletcher discuss the importance of this fundamental skill before entering higher education in computer science [3].

1.3 Purpose and limitations

In my thesis I will develop a proposition of a curriculum and assignments for teaching computational thinking to freshmen in computer science.

The following questions will be investigated:

q1 - What is computational thinking? How can it be described?

q2 - How is computational thinking taught and what is the main focus?

q3 - How is computational thinking used in education?

A discussion of the field of computational thinking, what the potential outcome of such a course would be and how a study of the field could be performed will be held. When discussing the impacts of computational thinking a special focus on women in computer science and how it could affect their interest in the field will be held. This discussion is important to the field since the absence of women in computer science is highly present [7].

The absence of studies in the field of computational thinking and the impact of teaching the skills calls for a pilot study. Unfortunately even a pilot study would take at least one year since the course would have to be designed, a trial set up and the test group and reference group would have to take at least one other course to be able to evaluate the effect of computational thinking skills. Due to time constraints I will only focus on the skeleton of the course.

The course will be designed using Umeå University as the current setting and will only focus on students at the bachelor program in computer science. The tasks will be briefly constructed, a proof of concept. The contents of the assignments will be limited to Artificial Intelligence and a general discussion on how to implement the concept of computational thinking into different subjects.

The chosen field is closely connected to abstraction, efficiency, algorithms, patterns, and decomposition - which are all qualities of computational thinking [3] - and is both easy to relate to and a popular field of computer science.

When performing the tasks and the curriculum, didactics, cognition and human learning will not be considered due to time and knowledge constraints. Neither will a comparison between computational thinking and other forms of thinking e.g mathematical thinking, logic thinking etc. be done.

1.4 Review of the Literature

The field of computational thinking is young. The movement of the field started in 2006 with Jeanette Wing's demand for teaching computational thinking to everyone everywhere and from there the it spread. The inspiration of the field spread but a clear definition of it was lacking. Most articles you find about computational thinking refer to Wings first article[4], using their own focus to give a statement of the field. Whit the new attention to the field the attempt to define and explain the scope and nature of computational thinking began. The scope and nature differs some between different authors but the agreement on the importance of the field led to the formation of a leader group from Computer Science Teacher Association(CSTA) and International Society for Technology in Education(ISTE) to develop a material for teaching computational thinking in the K-12 education. This material is then referred to by many articles describing computational thinking in practice, this together with material written by global companies such as Microsoft and Google. One can ask, are CSTA and ISTE free from influence by lobbyists? Can we rely on global companies for information on what and how to teach such proficiencies? In the articles one can read that you don't need a computer to learn computational thinking. With the involvement of computer corporations, one can question the effect on the importance of computers and programming in computational thinking.

Searching further for answers to the impact of computational thinking one ends up empty-handed. There seems to be no studies made even though the teaching of computational thinking has been put to work.

1.5 Method

Computational Thinking brought into the spotlight by Jeanette Wing in 2006 when she argued the importance of this fundamental skill for the twenty first century [4]. Since the field of computational thinking is so young there is a lack of information and specific studies in the field are hard to find. This demands a lot of effort into the search of relevant articles in the field. Since no large studies of the field or the outcome of the progress in the field was found I decided to perform a study of the literature of the field. To find information I searched for articles published in different journals, reports of seminars, theses, curriculums, online computational thinking education and information to teachers on how to integrate and work with computational thinking in the K-12 education.

My work was divided into four different parts.

1. Introduction, planning and Searching the field
2. Literature study
3. Construction of the proposal for the course
4. Discussion of the proposal and the field of computational thinking

When searching for literature I used the database IEEE, Umeå university library's database (www.ub.umu.se) and Google scholar (scholar.google.se). When searching for articles I wanted to focus on articles that explained the field and/or how it was brought into education. I especially tried to find research articles on the outcome of computational thinking in education. This was unfortunately not found in any notable amount. For keywords and search phrases see Appendix A.

The decision of which articles to read was based on the abstract and the headlines of the articles. I specifically searched for examples of implementation in education and definition of the field. Other articles that were read as references and deeper understanding of the nature of computational thinking were articles of computational thinking integrated to different fields and subjects.

To create my examples on how to introduce computational into the computer science education at the university a short study of the current curriculum for a bachelor of science in computing science with the main field computer science was needed to understand how my suggested course could be interpreted into the education. Next, I studied how computational thinking was integrated into the education by other schools and universities and tried to map their way into the Swedish university context. The examples were built without programming to match the prior knowledge of the students. Hambursch et al. said in their article [11] that it is important to teach computational thinking in a language familiar to the students. Since there is no formal requirement of knowledge in programming the assignment was built without any programming involved. To give the students a connection and the best user experience the assignments was to be related to the fields of computer science.

1.6 Vocabulary

Computation When talking about computation and computational problem I refer to problems that can be solved using concepts of computer science.

Women When talking about women I refer to women in computer science as a group.

K-12 K-12 is the American education from kindergarten up to the 12th year of education. In Sweden it would map to preschool up to the end of upper secondary school.

CSTA *Computer Science Teacher Association* is a membership organization that supports the computer science teachers and teachers of other computing disciplines. They are providers of opportunities for the teachers in K-12 to better understand the computing field and how it relates[13].

ISTE *International Society for Technology in Education* is a nonprofit organization serving its members of teachers and educators technologies, connections and ideas for connected learning[14].

2 Computational Thinking

The field of computational thinking contains many different parts, equally important and there is no generally accepted definition. After studying different opinions of the field and their definitions it could be summarized as:

> *Computational thinking is the process of solving a problem for automation with respect to aspects such as efficiency, economics and human behaviour.*

When facing a computational problem one has to understand the nature of the problem and its context. In the search for a solution a computer scientist has to consider the architecture, the user, the economical need, the structure and much more. It is a complex task which can easily lead to poor solutions if the developer has not been properly trained in problem solving. These fundamental problem solving skills can be summarized as computational thinking.

In 2006 the pioneer Jeanette Wing defined computational thinking as following:

> *Computational thinking is taking an approach to solving problems, designing systems and understanding human behavior that draws on concepts fundamental to computing.*

Since then the strive to define computational thinking has grown but a consensus on a precise definition is still absent. But since the riot for computational thinking started the definitions have agreed on a few characteristics. It seems to be reasonable to state that computational thinking is a mental tool for the process of solving a problem [12, 10]. This process can be described by decomposing a problem into smaller pieces to individually solve each smaller part. After the problem is decomposed the work with abstracting the problem to find the smallest solvable problem and to find the parts that can be drawn to a more general solution begins. When the problem has been abstracted and generalized the attempt to develop an algorithm to automate the solution begins. The algorithm must be efficient and easy to replicate and maintain. The solution must also consider organizing, modeling and simulating data in a proper way. When dealing with a computational problem it often requires collaborative work and therefore a proper language to describe the solution is needed. In computational thinking it is called the computational thinking language [3].

The process of solving problem using computational thinking can be divided into the following categories: 1) *Algorithms* 2) *Abstraction* 3) *Decomposition* 4) *Data processing* 5) *Reflection* 6) *Computational Thinking Language*.

Before we discuss these categories a short discussion of what computational thinking is not will be presented.

2.1 What Computational Thinking is Not

Computational thinking is not programming.
Computational thinking is not a way to solve a problem quickly.
Computational thinking is not a solution.
Computational thinking does not give a solution.
Computational thinking does not state what a good solution is.

2.2 Algorithms

An algorithm is a way of formulating and describing a precise method of doing things. Consider taking a trip with your car. Searching for a map online the generator provides the user with an road map describing when and where the driver should turn to find its destination. The road map is simply an instruction to the driver on how to drive, an algorithm for driving to the destination. Another example of an algorithm designed for humans is a recipe. When designing an algorithm for humans baking some descriptions can be left for the human to expound. For example the recipe does not tell the human how long to stir but rather says "mix well". If we wanted to translate the algorithm for automation using a computer we can not tell the computer to mix well. The algorithm has to be more precise.

In computer science the importance of a precise and definitive algorithm is crucial since the computer, in contrast to a human, will do exactly what the algorithm tells it to do and nothing else. The algorithm can therefore not say turn left in the roundabout but has to say take the third exit to your right. Coming back to the baking example the step of "mixing well" has to be more precise. For example stir the mixture for 2 minutes. The task is now precise and well defined.

When solving a problem using computational thinking the aim is to design an algorithm that is precise and well defined so that it can be automated using a computer [3, 10, 12, 15]. The algorithm enables control over the flow [3] serving the user to find test and evaluate the algorithm before it's even used. The algorithm should however not be suited for a specific language or architecture but general so that it can be implemented on any architecture using any programming language. Briefly put, an algorithm enables a general solution for automation [10, 3, 12].

Defining an algorithm for automation also gives us a way to simulate different situations so that we can organize and model data [10, 3, 12, 15].

Designing Algorithms

Amongst the first algorithms students tend to encounter seems to be the sorting algorithms. Such an algorithm can be derived both by practice but also by calculation. One such algorithm is *bubble sort*. This algorithm sorts a list by using pairwise comparison [16]. The *bubble sort* algorithm contains comparison, iterations, conditions and simple instructions.

2.3 Abstraction

Abstraction could be described as the art of taking the details out of a problem so that you can make a solution work in general. When facing a problem one must find what is relevant and what is not. After finding the real problem the hardest part of the abstractions begins with dividing the problem into different layers of abstractions and understanding how they all relate to each other [3].

Abstraction in computational thinking is finding the most suitable abstraction rather than one abstraction [12]. The importance of different layers of abstraction is highly relevant when working as a developer. When one has developed a system the system needs to be divided into different layers of abstraction to be explained to different types of user. Consider a website for instance. The end user needs not know what language was used, how the algorithms were implemented or what architecture it is currently running on. The content provider needs not know the architecture or the algorithms but rather how to add information and desired actions to the site. The maintenance person has to be fully aware of how to support the system and the algorithms but not what information is displayed.

> *An algorithm is an abstraction of a step-by-step procedure for taking input and producing some desired output.*
>
> — Jeanette Wing [12]

The use of abstraction is essential when creating algorithms. Without an abstraction there is a risk that the algorithm will cover unnecessary parts. Take a description of a recipe. If the recipe holds that you should stir with a "happy face" the algorithm might state that before we can stir we must find a happy face, even though it is not needed to make a good cake. [12]

In programming these layers of abstraction are often linked together by an application programming interface (API). The API needs to be designed with the edge of the abstraction in consideration. What information from this layer of abstraction is needed to communicate with a higher or lower layer of abstraction? Answering this kind of questions is not a matter of implementation but rather a question of the developer's ability of mastering computational thinking. When the interface of the abstracted layer is set the developer can choose how to implement it.

Wing states that the core of computational thinking is to define abstractions to work in different layers and truly understand how these interact and how they relate to each other. [12]

Working With Layers of Abstraction

A typical example of working with layers of abstraction is the development of applications for tablets or smart phones. Such a task involves different layers of abstraction. Think of a situation of paying for your bus ticket. Developing an application for this task demands the direct abstraction of the purchase. When abstracting the real world situation into a digital environment where the buyer should interact with the application the process of abstracting further into different layers starts. Imagine you have different teams responsible for different parts of the project. The

design is an abstraction of the interaction, the underlying functions an abstraction of the transaction and finally the confirmation is the abstraction of the exchange. The different layers have to consider different challenges and opportunities. The design must be appealing and built upon human behaviour and anticipations. It must also carry the interface to interact with the layer below, the underlying functions. The transaction has to investigate the reliability and security of the transaction of money. And it has to interact with the layer above for information on what to do and confirmation of the actions. Then it has to interact with the layer below, the process of buying and confirming a real life ticket to use.

2.4 Decomposing

Computational thinking provides a variety of strategies to tackle complex problems. One such strategy is to decompose the problem to reduce complexity in the process of finding a solution [5]. Decomposition could be described as the process of breaking a problem into smaller manageable parts [17]. In computer science this strategy is known as divide and conquer, an approach to divide the problem to smaller solvable parts. When working with computational problems such as creating an application or developing a new system decomposition is very important since it enables the work to be divided between different developers, departments or companies.

Decomposing a Problem

Studying the programming language Java the decomposition of the problem becomes obvious due to the structure of the language. When programming in Java the program needs to be decomposed into different methods and classes. The classes and methods can be seen as different granularity of the decomposition. The first decomposing derives the classes and within the class the structure is decomposed into methods and attributes.

2.5 Data Processing

When dealing with complex problems the need for processing data becomes obvious. A part of computational thinking is to determine what data is needed, decide how to represent it, the effect of visualization and the ability to analyze it. In short data processing is to give value to data, finding patterns and the ability to draw conclusions from data, collected and/or made available [17].

When outlining the real problem the question of what data is needed starts. Data about the context before the problem is even stated might be needed to avoid developing a product for the symptom and not the real problem. When data is collected a solid and proper representation must be considered. If we are dealing with physics or medicine we might end up with trouble in representing very large or very small numbers.

When producing data the proper visualization and analysis are critical. Providing the user with a set of random numbers without a proper structure is often of no good. The use of simulations and graphs becomes crucial. [17]

Working With Data

Today there is a lot of data in the running and how to make sense of it and how to make it available is an important skill. First, a good representation of data is needed. A common question when developing is how to represent infinity. When using a computer to analyze data a lot of data can often be derived even if only one value was requested. To return and visualize only the data needed could be very preferable. Without the ability to understand different forms of visualization such as graphs and simulations and when to use what data loses its value.

2.6 Reflection

One of the core concepts of computational thinking is reflection. Throughout the whole process of solving a problem reflection is used to ensure that the solution is, in fact a solution to the original problem. In the beginning the reflection should be upon defining the problem, to ask oneself what is really the problem. During the process the reflection continues when deciding methods and abstractions. But even after finding the solution the reflection does not stop. Reflection over the solution is important and questions like if the solution was in fact the solution for the problem that was given needs to be answered. Reflection over the solution must consider if the solution is economical, is the solution worth the cost of the development, is it profitable? The reflection also needs to consider human behaviour, does the solution stand in conflict with human behaviour and the anticipation of the user?[18]

Another part of reflection of the solution considers the efficiency [10, 3, 4, 5]. It seems to be agreed upon that efficiency is important not only due to speed but to resources as well.

Reflection Over the Solution

Evaluating a system for registering students the obvious question must be asked - “Can you register a student using the system?”. This question might seem odd but sometimes systems are developed with the real problem forgotten. A system might add student to a database, storing all their grades and merits. It might even contain all courses available but not the function of actually registering the student to a specific course.

The evaluation continues with the evaluation of the usability of the program, if it is safe and so forth. Then the question of efficiency starts. Not only considering storage and speed but the efficiency of the user. A system built for registering students might not be efficient if the administrator registering has to keep a lot of information in their own memory and go through several steps where information might be both lost and forgotten.

2.7 Computational Thinking Language

To be able to perform a computational process one must be able to express these thoughts and flows. If you can not express your thought the abstraction will be of

little use and the algorithms might end up poorly executed. In an attempt for a better understanding and use of the computational thinking concepts George Fletcher and James Lu introduced us to a concept of a computational thinking language. They compare the use and value of this language to other existing terminologies like the language to express mathematical concepts such as finding the *search space* and to be able to *derive* a simplification of a notation. [3]

A language for thinking computationally is important for the students to be able to understand information and to successfully discuss computational problems with others in the different fields of computer science. A common language or terminology also enables the student to overview ones solution and to have control of the data flow or the algorithms before the solution is implemented [3].

The computational thinking language however is not bound to a programming language but rather to computational concepts. To be fluent in the computational thinking language the student must master both the computational thinking vocabulary as well as the computational thinking notation [3].

Vocabularies

The terminology in computer science is a set of well defined words describing the process of computational thinking. These words need to be defined in a context similar to computer science but in a way so that the student understands them. The introduction of the words has to be strategic to the students knowledge of the different concepts [3].

When introducing *iteration* one can use multiplication as an example by letting the student perform the algorithm of multiplication as repeated addition with every plus sign as one iteration.

$$\begin{array}{cc} x*y, & x<y \\ y+y+...+y & i=x \\ x+x+...+x & i=y \end{array}$$

The student can then be introduced to the concept of *efficiency* by a discussion of how many iterations the two ways of multiplying require. [3]

When introducing the concept of *states* the student can be handed a set of different sentences and be asked which of the sets has the sentences in the correct order. Here the student has to consider time and consequences. When discussing the different states a *divide and conquer* concept can be introduced by asking the student to determine the correct order between two different states and then cancel out all the alternatives that contains these sentences in the wrong order. Here we can expand the definitions to also cover *dependency* where the student has to discuss if and in that case what the dependency between different states is [3].

Notation

The use of a proper notation can help the student to visualize the problem or its solution. For instance, the student could use a tuple[1] notation for structuring data

[1] A tuple can be described as an ordered list of elements. An n tuple is an ordered list of n elements. i.e $(a_1, a_2, ..., a_n)$

or different states. Combining this notation with a rewriting system[2] for visualizing state changes [3].

An example of the use of this notation is when defining a Turing Machine. The Turing Machine can be described as a tuple together with a set of transition rules, tuples within a rewrite system:

$$M = (\{q_s, r, w, b, e\}, \{0,1\}, \{0,1,\square,\$\}, \delta, \square, q_s, \{f\})$$

Here M is the notation of the Turing machine for write the reversal of the given input. The tuples $\{q_s, r, w, b, e\}$ describes the different states M can attain. If the current state is r the Turing machine is reading. The tuples $\{0,1\}$ and $\{0,1,\square,\$\}$ is the alphabet the Turing machine can read and write. δ is a set of rules for how the Turing machine can enter different states, read and write. Finally we notate the starting and accepting states. All these notations is a shorter writing for how a computer can read a word and revers it. If we were to describe the Turing machine in words it would take up a lot of space and the risk of misunderstanding is high.

Using δ we can shortly say that we have so called transition rules. These rules explains how we can read and decide what to do next. For the Turing machine M we get the following transitions for δ:

$$\begin{aligned}
(q_s, p) &\to (q_s, p, R)\\
(q_s, \square) &\to (r, \square, R)\\
(r, p) &\to (w, \$, R)\\
(w, \$) &\to (w, \$, R)\\
(w, p) &\to (w, p, R)\\
(w, \square) &\to (b, p, L)\\
(b, p) &\to (b, p, L)\\
(b, \$) &\to (r, \$, L)\\
(b, \square) &\to (e, \square, R)\\
(e, \$) &\to (e, \square, R)\\
(e, p) &\to (f, p, L)\\
(e, \square) &\to (f, \square, L)
\end{aligned}$$

Here we have the tuple (q_s, p) describing that we are in state q_s reading a p. The arrow tells us that when this happen this is the action to make. The tuple (q_s, p, R) tells us that we now stay in state q_s, we write a p and according to R move right. By just explaining parts of this powerful machine the compact notation helps us to get an overview of the process.

The notation of the solution can help the student to better understand the computational aspects of different problems. Different notations and how and what should be represented by different notations also forces the student to discuss granularity of the abstractions and how different data should be represented [3].

[2] A rewriting system can be described as a set of objects and relations on how to transform the objects.

When creating a solution to a problem a proper notation can give a more general solution. Fletcher and Lu present a problem with parsing and diagramming sentences as an example of how one can work with notation. Here the student has to find the different parts of a sentence. In the end the sentence is divided and notated in a general notation which enables the student to recursively create new sentences. [3]

By using notation to explain different rules for sentences the concept and notation can be used for the student to explore and use *recursion*.

$$\begin{array}{rcc} sentence & \rightarrow & noun-phrase verb-phrase \\ noun-phrase & \rightarrow & modifier noun | noun \\ & \vdots & \\ verb-phrase & \rightarrow & verb noun-phrase \end{array}$$

The notation for building a sentence can be derived as follow:

$$\begin{array}{rcc} (sentence) & \Rightarrow & (noun-phrase, verb-phrase) \\ & \Rightarrow & ((modifier, noun), verb-phrase) \\ & \vdots & \\ & \Rightarrow & ((the, summer), (is, over)) \end{array}$$

Here we use notations for recursively composing phrases. The notation enables us to show the derivation of the sentence of our choice. It is easy to follow and see that with just a shift we can produce another sentence, for instant we could have changed *"summer"* into *"winter"* and end up with a different sentence.

3 Computational Thinking in Education

The strive to make computational thinking a part of the basic skills alongside reading, writing and arithmetics prompted the teaching of computational thinking throughout K-12 [12]. The authors seem to agree that teaching computational thinking should start in the early years and develop during the K-12 experience [9]. But what role computational thinking should have and how it should be taught was not yet clear. The work to define computational thinking in the K-12 setting started in 2009 by the Computer Science Teachers Association and the International Society for Technology in Education. The focus of the project was to find the definitions and implementations of computational thinking in the given context. Questions of the curriculum outcome, standards and artifacts needed answering [9].

Computational thinking is not taught as a subject but rather integrated into the existing curricula alongside the nature and science subjects. The work benefits from the process of computational thinking by broadening the experience to involve more analytic work and a more agile process [9]. The discussion regarding computational thinking as a general subject, a discipline-specific topic or a multidisciplinary topic is not yet agreed upon [19]. There is still questions about whether computational thinking differs from other forms of thinking and therefore can be set as a subject of its own.

Computational thinking is also taught as a computer science course for science major students with the aim to teach the students tools they can use in their own field but also to increase the interest in computer science [11].

3.1 Computational Thinking in Practice

The authors in the field of computational thinking seem to agree that an important part of computational thinking is that it does not require a computer [12, 18, 20]. Computational thinking is used as a process of solving problems. The aim is to give the students the ability not only to use the tools for problem solving but also to create them. [9]

Grover(2013) reports that media is suggesting that computational programming should be a more common skill for everyone, adding algo**r**ithms as the fourth "r" alongside **r**eading w**r**iting and a**r**ithmetics as a 21st century skill. Computer science is added into the K-12 and similar curricula across the world and in UK a pilot program to teach computing has been launched. [19]

In her article "Bringing computational thinking to K-12" Barr reported the project to derive a set of goals for teaching computational thinking throughout K-12. She also suggested core concepts and abilities that the subject should fo-

cus on. The complete list of concepts and abilities can be found in Appendix C. Teaching computational thinking is to strategically teach the students to tackle any problem, easy or hard, and analyze and evaluate the process and solution. Teaching at a proper level and letting the student progress over time to tackle more complex problems will allow the students to gain confidence in solving different problems. These capabilities and concepts were then incorporated into different parts of the K-12 curricula [9].

Barr continues her article discussing the different concepts of computational thinking and how it can be integrated into different fields of the K-12 curricula. These concepts and activities and how it can be mapped into different subjects can be found in a table in Appendix B. This table show the diversity of the subject and how different parts of the computational thinking concept can be implemented and taught in different contexts and settings. The concepts of computational thinking can be taught in different subjects and transferred to other subjects. Barr reports that these concepts would benefit from increased use of the computational thinking vocabulary both among students and teachers. Not only is the language important but the environment that teaches the subject must change. The attitude of failed solutions and attempts must be changed toward acceptance amongst both student and teachers and more important the recognition of the significance of the analysis of an early failure that can put the student on the right track to finding the solution. Finally the work of the students should be carried out with the explicit use of decomposition, abstraction, negotiation and consensus building [9].

Lee(2011) reported in his article that students in the K-12 education work with modeling and simulation by simulating the spread of swine-flue in the school environment with different parameters. They also work with robotics where the students design and program a robot. Here they need to think about how the robot should interact and what factors they depend upon. The focus in this exercise is mostly on the abstraction where the students need to find a limited set of conditions the robot could react to. Automation also occurs when the students program the robots. Finally the students can work with game design and development. Here the focus is more on the analysis of the game; if their abstractions indeed are correct and efficient. The game also gives the students a chance to work with their abstractions and algorithms. When working with computational thinking a use-modify-create model can be applied to improve the progress and the students' feeling of autonomy over ones learning. The model is based on the students curiosity and understanding of the current device/program. When the students begin to understand the program they might want to modify it. For example change the colour. After a while when the students have gained more confidence and a better understanding of the system and underlying algorithms and architecture they might want to create a feature of their own. Here the progress of the knowledge is both clear and necessary [18].

Bringing computational thinking into the K-12 education material and exercises have been created. This material is a result of a cooperation between academia, national bodies and different organizations such as Google and Microsoft [19]. Exploring CS curriculum[1], a one year college preparatory curriculum for upper secondary

[1]http://www.exploringcs.org

students, and other material such as CS4HS[2] and Computing in the core[3] has been made available for teachers to join.

3.2 Computational Thinking in Higher Education

Introduction to Computational Thinking is a computer science course taught at the University of Purdue. The course was designed for non-major science students to fulfill the computational requirement of their education. The course was outlined by computer science teachers in collaboration with the science faculty. It was designed according to five guidelines [11]:

1. Lay the groundwork for computational thinking
2. Present examples in a language familiar to the students
3. Teach in a problem-driven way
4. The programming language should right away allow a focus on computational principles
5. Make effective use of visualization

Students are taught to formulate and solve problems by using abstraction, data structure and designing an algorithm to be satisfied computationally. Visualization and presentation are also skills the students are taught. For the students to better grasp the concepts the context of the learning has to be familiar. If you give a chemistry student a problem in the context of physics the student might not grasp the computational concept since the student has to focus on understanding the physics of the problem.

When learning computational thinking the students should be engaged in problem solving rather than following a step by step solution. When using a programming language the focus should only lie on the details needed for the student at the time. Other language specifics that are not important should be avoided.

The concept of visualization is important to a scientific context since it brings another level of evaluation to the science. Therefore the student has to master the visualization of the solution.

The course was given 15 weeks with two one-hour lectures and one two-hour lab per week [11]. The course outline can be found in Tabular 1.

The course was made for science-majors as a part of the requirement of a computational course in their education. The programming language of choice is Python since it allows the student to write advanced programs at an early stage using built-in packages. The course contained smaller programming exercises and four projects. The topics of the projects was 1) Manipulating Digital Audio 2) Computational Experiments on Percolation in Grids 3) Simulating Physical Systems 4) Analyzing Protein-Protein Interactions. The programming assignments are preparations for

[2] http://www.cs4hs.com
[3] http://www.computinginthecore.org

Table 1 Outline for Introduction to Computational Thinking

Basic Programming Tools (6 weeks)
Introduction to Python
Straight line programs, assignments to variables, type conversion, math library.
Conditionals and loop structures
Plotting using MatPlotLib and 3D visualization in VPython
Functions, parameters and scope. Recursion
Computational Tools and Methods (6 weeks)
Arithmetic and random numbers. Using NumPy. Examples of numerical stability and problem stability.
Introduction to simulations and Monte Carlo methods
Computational Physics: Ideal gas and Ising Spin simulations
adapting a generic Daemon algorithm and estimating parameters in a physical system.
Trees as a data structure, traversal and exploration.
Introduction to graphs, graph operations using NetworkX, graphs in science applications.
Bioinformatics: Modeling protein interactions using tree and graph representations. Visualizing graphs in Cytoscape and analyzing protein interactions using clustering techniques.
Grand challenges in scientific computing
Looking Under the Hood at Computer Science (3 weeks)
Object-oriented design. Use and design of classes, OO concepts. Dictionaries and spatial queries as examples.
History of computer science.
Limits of computing, intractability, computability.
Future models of computation: DNA computing, quantum computing.

the projects. The projects contained two parts, a programming part and an experimental part. The programming part is due earlier than the experimental part and if the student so chooses one can use the code from part one or code made available for the experimental part. The projects focus on representation of data and algorithms, manipulation of data, handling large data sets, dealing with round off error and overflow issues, computational complexity, iterations, different data types, recursion and visualizations. The students are always asked to visualize the computational results and write about their observations during the projects. [11]

One of the goals with this course is to increase the interest for computer science among the students. The aim is however not to turn science majors into CS majors. Harbrusch(2009) reports that when the course was given it had 13 students and the students were asked to answer a survey when entering the course answer it again when exiting the course. The survey showed that the student's interest in CS had increased after the course. The response from the students indicated that 60 per cent where interested in taking another course in computer science and 40 per cent plan to minor in CS. Feedback from the students showed that the problem

driven approach and the ability to quickly write meaningful programs were factors for increased interest. [11]

4 A New Approach for Teaching Computer Science to College Freshmen

In mathematics you are first taught arithmetics. You learn how to count, how to add and subtract. When you master these you learn how to multiply by repeated addition and division as a form of backward multiplication. When you have learned these basic forms of arithmetics you start to learn different ways of performing them and different algorithms that use them to solve more complex problems. No matter what task in the mathematics you are facing the problem will rest upon these basic concepts. The basics can be seen as the proficiency which is needed in all fields of a subject. Computational thinking as a process on how to solve a problem efficiently for automation with regards to the economics and human behaviour places computational thinking in the basics for computer science. No matter what field of computer science you are facing the proficiency of computational thinking holds to bring out a solution.

4.1 Computational Thinking as a Freshmen Course

The outcome of computational thinking has been discussed but not yet evaluated. According to Lu and Fletcher proficiency in computational thinking should be substantial before entering higher education in computer science. This proficiency would enable the student to easily perform problem solving and enable the student to shift focus from the process of problem solving to the specifications of the programming language and architecture. The outcome of such a course would also shift the focus from finding a solution for a specific problem to finding the *right solution* for the problem. Since computational thinking is not taught at upper secondary school in Sweden the students entering computer science studies have not necessarily been trained in systematic problem solving for automation. This course would also benefit other science minor and science majors since solving problems for automation is needed in many fields of science.

4.2 Syllabus

The course syllabus should be built on the principles of computational thinking with inspiration from the guidelines set up in practice. The course should be set in the context of the department of computer science at Umeå University to fulfill the requirements for a bachelor degree in computer science. To meet this context the course will be mapped to the local description for a degree of bachelor of science with the main field of study in computing science with specialization in computer science [21].

The course should focus on the particular process of using computational thinking together with the appropriate language to discuss and represent computational problems. Since the course is limited to five weeks there is not time to develop the skills for advanced problem solving. Therefore the course has to focus on the process of problem solving and the reflection upon it. The course given to science majors at the university of Purdue showed a clear focus on teaching computational thinking in a language familiar to the students. Since admittance to the bachelor program in computer science does not require knowledge of the field of computer science the course should not be taught with advanced and complex computational problems. However one can assume that the students share an interest in computer science and therefore the problems should relate to the field to increase the interest and make the transition easier. Another important aspect of computational thinking seems to be the ability to collaborate and evaluate both the progress of one's learning and abilities but also to discuss the solution of the problem with others. The following criterion ware collected from the degree requirement (The criterion in its original statement in Swedish can be found in appendix D):

1. The ability to, in both writing and speech, give an account of and discuss information, problem and their solutions with different groups.
2. Show insight in and understanding of terms and methods and their applications fundamental to computer science.
3. Show the ability to use theories and methods from computer science to solve problems in collaboration with others.
4. Show the ability to interpret the user requirements in programs and computer system in the context of social, economical and cultural aspects and values.

These requirements outline the ability to discuss and solve problems with relation to computer science in collaborative environments. These are relevant to the concept of computational thinking since it requires the ability to abstract, decompose problems, form algorithms and discuss the efficiency, relevance and value of the solution. The use of a computational thinking language is also implied in the requirements above.

Contents

The course should focus on teaching the process of problem solving outlined by computational thinking definition and discussion. The course should present different terms and notations from the computational thinking language(CTL). These terms should then be defined and put into context of use. The students should be presented to a problem where they collaboratively can practice the meaning of the term. For instance, if the term *algorithm* is presented to the students the tutor should define the word and show small examples of how it can be used and show the connection to automation and technical artifacts. The process of solving problems should focus on *algorithms*, *decomposition*, *abstractions*, *data collection, patterns, data representation* and *data analysis*. Another important part of computational thinking is the quality of the solution. Here important concepts such as *efficiency, economics, iteration, recursion* and *human behaviour* must be considered and introduced. The course should also provide the student with training in discussing and collaboration with others.

Expected Learning Outcomes

Computational thinking requires skills for defining a problem and formulating a proper question to answer. When one is faced with a real world problem the first step towards an efficient solution is to define a problem for automation. It might sound as an easily learned ability but it requires a deeper understanding of abstraction, data analysis and algorithmic thinking which places it right in the front for computational thinking.

When learning the concept of computational thinking the ability to actually solve a problem is not interesting. Indeed should substantial practice in computational thinking lead to improved ability to solve a problem but the process of how to solve a problem and reflect upon the quality of the solution is more important. The ability to account for the process gives the student the tools to reflect upon one's strategies and therefore improve it with a lifelong practice. The student should however not only be able to account for the process but also demonstrate how it can be used.

One of the cornerstones of computational thinking is abstraction. It is only through abstraction the complex problem of real life can be transferred to a simple algorithm for a technical device to process it. To understand the meaning of abstraction and how to decompose a problem in different layers of abstraction and truly understand how these different layers of abstraction relates to each other is highly important when dealing with computational thinking.

When tackling a problem handling data is most likely a part of the solution to understand how data can be processed and how we can use automation to represent and analyze and then in an appropriate way visualize it. Without the ability to handle data many problems might end with a solution no one can understand.

Computational thinking is not about finding a solution, it is about finding a good solution. A solution that is not efficient, nor economical and does not meet the human desire nor is designed for human behaviour might not be of any use. But to find a good solution is not easy, in some cases it is not even possible, so the ability to discuss the quality of the solution with regards to different factors becomes highly important.

Required Knowledge

When searching for information in the field of computer science there is one dominating language for information, namely English. There is a saying that you become good at what you practice. Since you most likely has to find information in English the benefits of learning the process of problem solving, the terminology of the field and the confidence in communicating in an international language should be clear. However, since this should be a course given to college freshmen it might be intimidating for the student to dive into a new field in a foreign, but not unfamiliar language. One can also reason that since the information available is most likely in English it would become a false security to be taught in Swedish. Therefore I would recommend the course to be given in English and substantial proficiency in English should be required.

No knowledge of computer science should be needed since the benefits of computational thinking are not limited to computer science.

Form of Instruction

When working with computational thinking the importance of progress is obvious. When teaching computational thinking in the K-12 setting the progress is shown over time during the 12 years of education. The student first meet problems with low levels of abstraction and shorter algorithms and as their abilities and knowledge increase the problems gets more complex and the layers of abstractions start to show. Since the freshmen course is to e given in such a short time(5 weeks) this progress must be made from the teachers introduction to the students exercises and assignments. When introducing a new concept it is beneficial to use an everyday, easy to relate to problem and from there step by step make the concepts more advanced. The student should work with problems with clear progression in a problem-driven environment. To teach in a problem-driven manner encourages the student to reflect and discuss since the solution is not given.

During the exercises the students should be divided into groups and collaboratively solve the problems given. A tutor should be assigned to the group to help the group use the computational thinking language and to move the discussions forward if it stagnates. The tutor should not be afraid to help the student forward if needed since the reflection on the problem solving and the discussion of how and why different strategies should be used is more important. ISTE and CSTA reports in their material for teachers that the questions of how and why give the student the strategy to continue the process.

When introducing the students to a new concept the tutor could use the use-modify-create strategy. The use-modify-create strategy let the students use a tool which eventually can be modified by the students. When the students has progressed and gained knowledge of the tool and the underlying design the students can create their own modification to the tool. Using this pedagogy ensures the students progress in their learning since the understanding of underlying algorithms and structures must be clear to the student before the step of modifying different parts of the tool. Furthermore the student has to fully understand the different layers of abstraction, how they relate, what architecture the solution is built upon and how the algorithm works before one can create a piece of ones own. The student must also be able to analyze the effect of the added piece.

After substantial practice in the different parts of computational thinking and training in the computational thinking language the students should be divided into groups and demonstrate the process of computational thinking based on a real world problem of their choice. Here the tutor's role is to see that the problem of choice is not too big or complex for the students to tackle. However, the students do not have to find a solution to the problem as long as they reflect upon the process and decompose the problem and abstract it in a way so that it is clear for the tutor what the students need to know before solving the problem. Due to time one can not expect the students to actually investigate different areas in the process of finding a solution.

Examination Mode

To grade ones ability to solve a problem seems to be out of the question for computational thinking since it is not a way to find a solution but the ability to find

the right solution, if one can be found. Since mastering this proficiency is a life long goal the ability to reflect upon the process and to become fluent in the language seems more relevant. Therefore the examinations be focused on participation in exercises and the reflection of the exercises and process of solving problems along with the use and understanding of the computational thinking language.

4.3 Assignments

Studying the practice of computational thinking makes it obvious that the practice of the tools is the most common way to teach computational thinking. Computational thinking as a subject is not yet taught. Since this course should focus on the tools rather than the practice the assignments should focus on reflection and analysis of the tools and one's work. One of the qualities in computational thinking is the ability to collaborate and reflect upon one's abilities and effort. Since computational thinking is not a ready to use solution to a problem but rather a tool for you to develop a strategy for finding a solution the assignments ought to be problem-driven.

Since computational thinking is a multidisciplinary tool the field of the problem is not important. In the K-12 education computational thinking is taught within subjects such as language arts, science, social studies and computer science. When finding a problem to solve, try to find a problem that can be decomposed, that contains different layers of abstraction, where data collection, representation and visualization is preferable.

When the student has worked with computational thinking on smaller problems and analyzed the process and is familiar with the computational thinking language the progress should involve a more complex problem. Letting the student pick a real life problem to analyze and process. The students pick a problem with guidance from the tutor. This project should be presented with a project diary where the students write how the problem can be solved using the concept of computational thinking. The process should be described in the computational thinking language whenever possible. The students should also analyze and discuss what variables must be considered when finding a solution with regards to economics, efficiency and human behaviour. The students should also give a suggestion on how to evaluate the quality of the solution.

The assignment can be found in Appendix F.

5 Discussion

When creating a course for college freshmen there is a fine line between silliness and devastation. For some the introduction course is almost a joke and for some it is the hardest thing they have done in life. The media constantly reports a big difference in the results between different schools and this is a hard problem to face. The university has to meet the students where they are and therefore the course has to challenge both students with good and poor knowledge of the upper secondary curriculum. With the course proposed I hope to meet the students at their level since the concepts of computational thinking can be set in a great variety of problems. The focus of my course is on the reflection upon the process of problem solving using the concepts of computational thinking. My hope is that the students will learn how to systematically solve problems and to reflect upon the process and factors that affect the requirements of the solution. The course was built to imitate the progress and focus of computational thinking in practice. I have not yet found a study to show the outcome of bringing computational thinking into the K-12 experience and therefore my conclusion of the outcome can only be based on the hypothetical outcome proposed by the different authors. If their propositions are correct the course could give the students a leap into computer science and the ability to master the 21st century requirements. For higher education Fletcher an Lu propose that the result will increase when the students' focus is shifted from the process of solving problems to the limitation and advantages of the tools and artifacts in use.

From the result at the university in Purdue the interest in computer science could indeed increase if the course was offered to other science-majors or science-minors. The concepts of computational thinking is indeed multidisciplinary and we could get an increased interest in minors in computer science. My hope is that the analytic and problem solving focus of the course will appeal to women and therefore increase the interest in computer science amongst them. The old saying that computer science equals programing could hopefully be broken when introducing the problem solving, interactive and social part of computer science occupations.

5.1 Validation of the Conclusion

Since the course is developed with the guidance of the definition of computational thinking and how it is performed there is no validation of the conclusion. Since I have not found any studies made of the outcome of teaching computational thinking there is no validation of the theories either. However I reason that the course proposed could be used for validation. I would propose a qualitative study where you have two groups of students. A reference group which take the course as their first course at computer science and one control group that does not. After their freshman

year the students' grades are evaluated. An interview with the students of their experience of if the course actually helped them during their first year and if so, how and why. Repeating this for three years interviewing three different batches of students collecting data would give a suggestion of the impact. The questions to answer when analyzing the data collected should be:

1. Can we find patterns in the result?
2. How much does the attitude towards the impact of computational thinking differ?
3. Can we find patterns or trends in the answers of if and how computational thinking is helpful?
4. Can we find patterns or trends in why computational thinking did help?
5. Did the reference group show better grades than the control group?
6. Did the reference group have a better learning experience through their first year?

5.2 Further Work

A clear and precise definition of computational thinking is needed for the growth and survival of the field. Indeed this is an important field since our society becomes more and more digital.

When a clear definition is at hand the work of validating the impact of computational thinking in education needs to be initiated. My suggestion is to investigate the impact of computational thinking in the field of computer science to determine if computational thinking is actually the problem solving of computer science. If there is no impact in computer science then why should we believe that it has impact anywhere else? After determining the impact of its own field the work with validating computational thinking as a 21st century skill for everyone everywhere must be determined.

It is my true belief as a digital native that this is indeed a skill for the future but the lack of validation is somewhat shocking.

Bibliography

[1] Wikipedia. Outline of computer science. http://en.wikipedia.org/wiki/Outline_of_computer_science (visited 2014-04-30), 2014.

[2] Umeå Universitet. Kursplan - programmering i c 7 5hp. http://www.cs.umu.se/utbildning/kurser/kurs/?currentView=syllabus&code=5DV114 (visited 2014-04-29), 2014.

[3] James J Lu and George HL Fletcher. Thinking about computational thinking. In *ACM SIGCSE Bulletin*, volume 41, pages 260–264. ACM, 2009.

[4] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[5] David Barr, John Harrison, and Leslie Conery. Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, 38(6):20–23, 2011.

[6] SCB Universitetskanslersämbetet. Higher education. students and graduates at first and second cycle studies 2012/13. http://www.uk-ambetet.se/download/18.32335cb414589905b288a/studenter-examinerade-grund-avancerad-niva-2012-13-SM-2014-01.pdf (visited 2014-05-01), 2014.

[7] Christie Lee Lili Prottsman. *Computational thinking and women in computer science*. PhD thesis, University of Oregon, 2011.

[8] Eric S Roberts, Marina Kassianidou, and Lilly Irani. Encouraging women in computer science. *ACM SIGCSE Bulletin*, 34(2):84–88, 2002.

[9] Valerie Barr and Chris Stephenson. Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1):48–54, 2011.

[10] National Research Council. *Report of a Workshop on the Scope and Nature of Computational Thinking*. National Academies Press, 2010. http://www.nap.edu/catalog.php?record_id=12840 (downloaded 2014-04-08).

[11] Susanne Hambrusch, Christoph Hoffmann, John T Korb, Mark Haugan, and Antony L Hosking. A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin*, 41(1):183–187, 2009.

[12] Jeannette M Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.

[13] CSTA. About iste. http://csta.acm.org/, visited (2014-05-26), 2014.

[14] ISTE. About iste. https://www.iste.org/about-iste, visited (2014-05-26), 2014.

[15] John F Sanford. Core concepts of computational thinking. *International Journal of Teaching and Case Studies*, 4(1):1–12, 2013.

[16] Wikipedia. Bubble sort. http://en.wikipedia.org/wiki/Bubble_sort, visited(2014-05-23), 2014-05-23.

[17] ISTE, CSTA, and NSF. Computational thinking teacher resources, 2011. http://csta.acm.org/Curriculum/sub/CurrFiles/472.11CTTeacherResources_2ed-SP-vF.pdf.

[18] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. Computational thinking for youth in practice. *ACM Inroads*, 2(1):32–37, 2011.

[19] Shuchi Grover and Roy Pea. Computational thinking in k–12 a review of the state of the field. *Educational Researcher*, 42(1):38–43, 2013.

[20] George HL Fletcher and James J Lu. Education human computing skills: rethinking the k-12 experience. *Communications of the ACM*, 52(2):23–25, 2009.

[21] Umeå Universitet. Lokal examensbeskrivning. http://www.umu.se/digitalAssets/10/10510_filosofie_kandidatexamen_inr_datalogi.pdf (visited 2014-05-05), 2014.

A Keywords and search phrases

When searching for articles the following keywords was used:

- Computational Thinking
- Education
- Curriculum
- Computer Science
- K-12

And the following search phrases was used:

- “Computational Thinking”
- “Computational Thinking in education”
- “Computational Thinking curriculum”
- “Computational Thinking in computer science”
- “Thinking like a computer scientist”
- “Studies of Computational Thinking”
- “Effects of Computational Thinking”
- “Teaching Computational Thinking”
- “Computational Thinking at university”
- “Computational Thinking science”

B Concept and Abilities

In this appendix the core concepts and abilities for computational thinking in the K-12 education are presented. The original table was collected from *Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?* [9]

CT Concept, Capability	CS	Math	Science	Social Studies	Language Arts
Data collection	Find a data source for a problem area	Find a data source for a problem area, for example, flipping coins or throwing dice	Collect data from an experiment	Study battle statistics or population data	Do linguistic analysis of sentences
Data analysis	Write a program to do basic statistical calculations on a set of data	Count occurrences of flips, dice throws and analyzing results	Analyze data from an experiment	Identify trends in data from statistics	Identify patterns for different sentence types
Data representation	Use data structures such as array, linked list, stack, queue, graph, hash table, etc.	Use histogram, pie chart, bar chart to represent data; use sets, lists, graphs, etc. to contain data	Summarize data from an experiment	Summarize and represent trends	Represent patterns of different sentence types
Problem decomposition	Define objects and methods; define main and functions	Apply order of operations in an expression	Do a species classification		Write an outline
Abstraction	Use procedures to encapsulate a set of often repeated commands that perform a function; use conditionals, loops, recursion, etc.	Use variables in algebra; identify essential facts in a word problem; study functions in algebra compared to functions in programming; Use iteration to solve word problems	Build a model of a physical entity	Summarize facts; deduce conclusions from facts	Use of simile and metaphor; write a story with branches
Algorithms & procedures	Study classic algorithms; implement an algorithm for a problem area	Do long division, factoring; do carries in addition or subtraction	Do an experimental procedure		Write instructions
Automation		Use tools such as: geometer sketch pad; star logo; python code snippets	Use probeware	Use excel	Use a spell checker
Parallelization	Threading, pipelining, dividing up data or task in such a way to be processed in parallel	Solve linear systems; do matrix multiplication	Simultaneously run experiments with different parameters		
Simulation	Algorithm animation, parameter sweeping	Graph a function in a Cartesian plane and modify values of the variables	Simulate movement of the solar system	Play Age of Empires; Oregon trail	Do a re-enactment from a story

Table 2: Core Computational Thinking abilities and concepts

C The Core Concepts in Context of Capabilities, Dispositions, Pre-dispositions and Classroom Culture

This is the list of core concepts that were discussed by the participants of the project on how to bring computational thinking into the K-12 experience. The list is collected from the article *Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community?* written by Barr et al:

Concepts

- Design solutions to problem (using abstraction, automation, creating algorithms, data collection and analysis)
- Implement designs (programming as appropriate)
- Test and debug
- Model, run simulations, do system analysis
- Reflect on practice and communicating
- Use the vocabulary
- Recognize abstractions and move between levels of abstractions
- Innovation, exploration, and creativity across disciplines
- Group problem solving
- Employ diverse learning strategies

Abilities

- Confidence in dealing with complexity
- Persistence in working with difficult problems
- The ability to handle ambiguity
- The ability to deal with open-ended problems

- Setting aside differences to work with others to achieve a common goal or solution
- Knowing one's strengths and weaknesses when working with others

D Degree requirements

Presented in this appendix is the chosen degree requirement for the Bachelor of Science in Computer Science in their original language.

1. Visa förmåga att muntligt och skriftligt redoföra för och diskutera information, problem och lösningar i dialog med olika grupper
2. Visa insikt om och förståelse för datavetenskapliga begrepp och metoder och hur de tillämpas
3. Visa förmåga att i samverkan med andra kunna använda datavetenskapliga teorier och metoder för att lösa problem
4. Visa förmåga att tolka användarkraven på program- och datorsystem i kontexten av sociala, ekonomiska och kulturella synpunkter och värderingar

E Syllabus

Computational Thinking 7,5hp

Credit Points 7.5 Credits
Level Basic Level
Main Field of Study and Progress Level Computing Science: first cycle, has less then 60 credits in first-cycle course/s as entry requirements
Grading Scale Pass, Fail

Contents

The course provides skills in problem solving using the concepts of computational thinking. The course trains the student to the problem solving process by using decomposition, abstraction, writing algorithms as well as accounting for dependency, efficiency and quality of solution. The course provided skills in representing and analyzing data to draw conclusions. The use of computational thinking language provides the student with a vocabulary and notation for describing and understanding computational problems and solutions. The students are trained in formulating problems and finding solutions for automation, designing algorithms with regards to efficiency, economics and human behaviour. A focus on solving problems, reflecting on and analyzing the process of finding a solution will be held.

Expected Learning Outcomes

Ability to formulate a problem for automation from a real life example
Account for the process of problem solving using computational thinking
Demonstrate problem solving using computational thinking
Ability to understand and use computational thinking language
Ability to abstract problems to find solution and confidence to handle different layer of abstraction and understand how they relate to each other
Ability to represent and analyze data, find patterns and draw conclusions
Ability to reflect upon a solution, the process and one's abilities
Ability to discuss efficiency, dependency and the quality of the solution

Required Knowledge

Proficiency in English equivalent to Swedish upper secondary course English A (IELTS (Academic) with a minimum overall score of 5.5 and no individual score below 5.0. TOEFL PBT (Paper-based Test) with a minimum total score of 530 and a minimum TWE score of 4. TOEFL iBT (Internet-based Test) with a minimum total score of 72 and a minimum score of 17 on the Writing Section).

Form of Instruction

Education consists of lectures introducing the different concepts, problems, vocabularies and notations. Mandatory seminars where students collaborate to solve and discuss different problems and analyze the quality, efficiency and different aspects of the solution. Individual assignments based on the seminars. A collaborative project.

Examination Mode

To pass the course the student has to actively participate in 4 seminars and hand in 4 individual assignments based on the exercise during the seminars. The student also has to participate in a collaborative project and hand in a written report.

The student can only be given pass or fail. If a student fails an assignment a re-exam will be held. If a student receives fail at a seminar the student will, if possible, be offered a re-seminar or a written re-exam.

F Exercises and Assignments

F.1 Artificial Intelligence - The maze problem

The students are divided into groups of 4-5 students. The students are given a grid with different obstacles drawn inside simulating a maze. The maze should have at least three different ways to exit. Each group should have a tutor to help it use the computational thinking language and guide it when needed. The tutor should, if possible, just observe and only interact when process stagnates.

Instructions

1. Write step by step directions for the Robot to exit the maze.
2. Write three different directions for the Robot to exit the maze.
3. Discuss which one of the different directions is most efficient and why.
4. Discuss if you can give directions without knowing what the maze looks like? What information is then needed?
5. Design an algorithm to exit an unknown maze.
6. Discuss what impact a moving object within the maze would have.
7. Design an algorithm for exiting a maze with moving obstacles.
8. Compare your algorithm with another group's algorithm and discuss the qualities of the algorithms.
9. Optimize the algorithm.

Computational Thinking Language

Terms to be used during the exercise and assignments. Other useful terms might be added.

- Abstraction
- Algorithm
- Iteration
- Ambiguity
- Efficiency
- Generalization

- Automation
- Complexity

The tutor should explain these terms and encourage the student to express and use them.

Assignment

Let the student write a discussion and reflection of the exercise. Ask the student to explain the different keywords and how they related to the exercise. Let the student describe the process of solving the problems given and analyze the solutions with regards to efficiency and human behaviour. Examples of questions the student could be asked is:

- Formulate the real problem of the robot.
- Describe the process of solving the maze problem
  - What strategies did you use?
  - How did you evaluate your solution?
- Use the Robot-problem to describe the keywords.

Let the student reflect upon the process and suggest improvements in the problem solving strategy. Use these ideas of improvement as anonymous material for group discussions and let the students discuss the impact of the improvements on problem solving.

F.2 Computational Thinking Project

The students should be divided into groups of 3-4 students. Recommended time limit 1 week.

Instructions

Let the students choose a real life problem to solve. Then let the students try to find a strategy using the concepts of computational thinking to solve the problem for automation. Each group should have an assigned tutor to turn to for guidance. The students should document the process of solving the problem and what information or data is needed to find the solution.