Computational Thinking: Model, Pedagogy, and Assessment

Austin Cory Bart
Computer Science,
Virginia Tech,
Blacksburg, VA,
acbart@vt.edu

December 17, 2013

1 Abstract

Computational Thinking has grown in importance and relevance in education, and the time has come for it to be unilaterally added to undergraduate curriculums. In this paper, we begin by describing the history of Computational Thinking and the research undertaken on the matter in the Computer Science Education community. Next we attempt to identify the exact operational definition of Computational Thinking. Finally, we discuss a potential measure to teach and assess understanding through a design-based research project that will iteratively build up a novel course and accompanying workshop. The expectation of this project is that, by working with domain experts, computational thinking experts, and education specialists, great changes could be made in the understanding of all students at the university level.

2 Introduction

In the past few years, there has been a push for "Computational Thinking" in curriculums outside of Computer Science. The phrase was coined by Seymour Papert in 1993 [19] and popularized by Dr. Jeannette Wing's 2006 paper [24], which opened a floodgate of discussion about the term. Unfortunately, there is still limited consensus on **what** exactly CT is, whether it should be universally taught, how it should be taught, and how to identify when it has been taught.

An excellent resource for summarizing the history of Computational Thinking research is the 2013 dissertation by Wienberg [22]. This comprehensive survey analyzed 6906 papers directly or indirectly related to Computational Thinking from 2006-2011, describing research efforts and findings. Those papers were filtered down to 164 papers that substantially related to Computational Thinking in order to draw more meaningful conclusions. Finally, 57 of these papers were given closer treatment to analyze their research methods.

Weinberg paints an interesting picture of Computational Thinking research. As might be expected, there has been a steady increase in published papers on the topic since 2006. The movement is focused in the US, with less than a quarter of first-authors being outside the United States, even though 55% of Computer Science research (beyond just Computer Science Education research) is published outside of North America [21]. Research is divided evenly between K-12 and undergraduate, and there is no strong presence of research on continuing or post-graduate education initiatives that involve Computational Thinking.

An important contribution by Weinberg is a taxonomic breakdown of the aforementioned 164 research papers based on whether they primarily focus on modeling, pedagogy, or assessment. This taxonomy is documented in Appendix A, and a summary of the results of this breakdown is shown in Figure 1. Over half of the research on CT describes approaches to pedagogy (Curriculum and Program Description), leaving a small amount to modeling (Philosophy and Opinion) and assessment (Research and Evaluation). The lack of assessment research is understandable given the youth of this area of research, but still troubling.

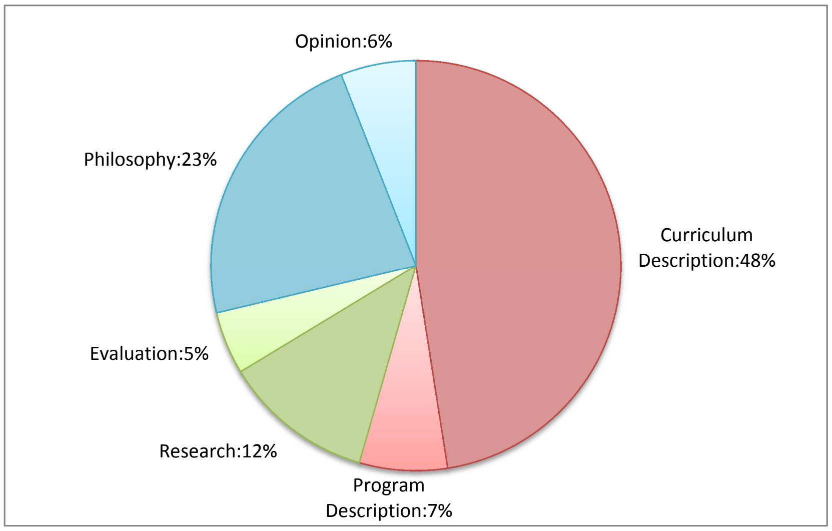

Figure 1. Percentage of Papers in Each Category of the Weinberg Taxonomy

Even more troubling, however, is the further analysis of the 57 empirical studies. Only fifteen (26%) studies include or sought an operational definition of computational thinking, and only six go beyond the superficial (solely describing computational thinking as a ”way of thinking”, a ”fundamental skill”, or a ”way of solving problems”). The failure to identify an operational definition weakens the theoritical strength of the studies. This weakness likely stems from the background of the researchers: 28% of the articles involved non-CS majors and **only 18% of the articles involved education experts.** In other words, over four-fifths of this educationally-oriented research was performed by people with no real formal training in educational research techniques. This is particularly troubling given that Computational Thinking is a strong target for interdisciplinary endevaors.

Weinberg reflects on the continuing debate about the importance of Computational Thinking:

> Many, like Wing, believe computational thinking to be a revolutionary concept, one as important to a solid educational foundation as are reading, writing, and arithmetic (Bundy, 2007 [2]) (Day, 2011 [7]). Others believe its potential and significance are overstated (Denning, 2009 [8]; Hemmendinger, 2010 [12]), and some have voiced concern that by joining

forces with other disciplines computer science might be diluting either one or both of the participating disciplines (Cassel, 2011 [5]; Jacobs, 2009 [15]). Both the praise and the criticism for computational thinking could perhaps be tempered by reflecting on a historical quote by Pfeiffer in 1962: Computers are too important to overrate or underrate. There is no real point in sensationalizing or exaggerating activities which are striking enough without embellishment. There is no point in belittling either. (Pfeiffer, 1962 [20]).

This paper will take as a given that Computational Thinking has a place in higher education. Instead, it will begin by attempting to reconcile the dominant definitions of Computational Thinking into a unified form. Then it will propose a revolutionary, wide-spread program modification that will introduce Computational Thinking to undergraduates, building on modern educational theory and existing research where applicable. Finally, potential assessment measures will be discussed, along with a plan for further development of these measures. The distribution of content in this document loosely follows the distribution given in Weinberg's taxonomic breakdown previously summarized in Figure 1.

3 Proposed Model

In order to research and investigate Computational Thinking, it is necessary to come up with an operational definition. There are many options available from the literature, of varying degrees of complexity and validity. As previously mentioned, the original paper by Wing did not define Computational Thinking, which has led to arguments about its exact nature. In this section, these arguments will be reconciled into an abstract model (dubbed **”The General Unified Model of Computational Thinking”**), which can then be used to instantiate ”sub-models” for various academic domain (e.g., Biology, Political Science, Business, etc.).

3.1 Dominating Definitions

Since 2006, a number of organizations and research groups have attempted to create definitions for Computational Thinking, with varying success. An important attempt was the 2009 National Research Council's Workshop on Computational Thinking, which brought together a large team of Computer Scientists in order to explicitly define the term. Unfortunately, no strong consensus was achieved, although a rich discussion occurred with a number of valuable conclusion.

Fortunately, subsequent efforts have been more helpful in generating an exact definition. We review four of these definitions that have had strong endorsement for one reason or another. In the interest of efficiency, only the first definition will be fully explained, with the remainder only being expressed in terms of their contrast. For the complete definitions of the remainder, they are conveniently listed in Appendix B.

3.1.1 Google Definition

The simplest of the four definitions chosen is the one espoused by Google. As one of the largest and most powerful technical companies in the world with a strong moralistic agenda, Google has a strong share in improving education for computationally-enabled workers. In fact, their definition is preceded by their commitment to ”promoting computational thinking throughout the K-12 curriculum” [14]. They identify only four key characteristics of Computational Thinking:

1. Decomposition Breaking down a problem to enhance clarity.

2. Pattern Recognition Noticing similarities or common differences to make predictions or shortcuts.

3. Pattern Generalization and Abstraction Filtering out information that is unimportant for the context and generalizing the information where necessary.

4. Algorithm Design Developing a step-by-step strategy for solving a problem.

This definition was chosen as the base because it only contained 4 ideas, with the expectation that the other definitions would completely subsume this definition. These four characteristics should be seen as skills that a Computational Thinker should possess.

3.1.2 Vs. Computer Science Teachers Association (CSTA)

The Computer Science Teacher Association not only created an operational definition of CT, but also performed a large-scale survey of Computer Science educators in order to measure its validity. They were able to achieve ”overwhelming” consensus on the 11 elements they identified [9].

The first half of the list is similar to the Google definition, with a few exceptions. First, Problem-Formulation and a computer-based approach is explicitly mentioned, going beyond the idea of Decomposition (where that problem is broken down into manageable pieces). Additionally, Data Organization is described in addition to Data Analysis, which seems like a necessary antecedent to Pattern Recognition or Generalization. One novel feature is the idea of building an algorithmic solution with the intent of optimizing some aspect (e.g., time, space, energy, exertion etc.). From these two views alone, we start to see the concepts of Problems, Data, and Processes start to be treated in similar ways: developed, organized, analyzed, broken down, abstracted, generalized, and a host of other verbs.

The second half of the list, however, adds an entirely new dimension to Computational Thinking: the attitudes and dispositions that a Computational Thinker should have. This more Humanistic aspect goes beyond the Cognitivist mentality of the Google definition. It is important to note that the assessment measures for these will be radically different. The new elements are: confidence in dealing with complexity; persistence in working with difficult problems; tolerance for ambiguity. The last two items in the list (the ability to deal with open-ended problems and communicate with others) are, despite the definition, better identified as being skills rather than attitudes.

3.1.3 Vs. AP CS Principles

The CS Principles project is a national initiative to teach Computational Thinking at the high school level through an Advanced Placement class. The curriculum has already seen wide-spread adoption at roughly 40 high schools and 10 colleges, as of June 2013 [1]. This project is actually one of the most explicitly defined, with a host of course objectives of varying detail. The particular level of detail that we focus on is near the top: the six ”Computational Thinking Practices”.

The first Practice is so far incompletely represented in our evolving definition, the idea of ”Connecting Computing”. This skill entails that students should not only be able to generalize concepts of computing and recognize patterns, but should also be able to identify the impact of computers on society and the ways that humans and computers interact with each other. They further emphasize the communicative aspect (”Communicating”) by requiring that students use ”accurate and precise language, notations, or visualizations”. Ultimately, they require that communication be localized too: ”Collaboration” must occur in the classroom, both to solve a problem and create an artifact. The idea of creating ”Computational Artifacts” is another new concept, one that extends the idea of Computational Thinking in an interesting direction.

3.1.4 Vs. Wing, Snyder, and Cuny

Although Wing never gave a definition in her original 2006 paper, a follow-up paper in 2010 cited a collaborative definition with Snyder and Cuny that identified what a Computational Thinker should be capable of. This definition covers many of the concepts previously seen, but gives special weight to identifying the powers and limits of computational tools when applied to a problem.

3.2 The General Unified Model of Computational Thinking

Figure 2 represents the completed General Unified Model of Computational Thinking. In order to draw together these varying definitions of Computational Thinking, we applied a Humanistic, Socio-Cognitivist perspective. This enabled us to build a hierarchal model, categorizing our components into Hard Skills, Soft Skills, and Attitudes. An expansive mindset was used when deciding what to include: we seek to have as broad a definition as possible, in order to effectively capture a wide range of subjects encompassed by Computational Thinking. As an Operational Definition, each leaf in the hierarchy represents a skill or attitude that a Computational Thinker exhibits. When evaluating a pedagogical tool or student’s learning, these element are not required in totality; some subset must be present, but much leeway can be given in how items reflect these traits.

This definition should be considered open-source: it is not intended to be an end-product, but an extensible and modifiable artifact that can be re-evaluated in light of new arguments and evidence. Furthermore, it is meant to be contextualized for a given disciplinary domain: Computational Biology should be expected to have a different instantiation of the model than Computational Law, but both of these should descend from the General Unified Computational Thinking model. An inheritance model could potentially be described as occurring between Colleges and Disciplines; you’d expect certain common behavior among Engineering Domains (and then further differentiation between, say, Chemical and Environmental Engineering) that is distinct from those found in the Humanities. In order to develop these individualized models, domain experts should be consulted.

3.3 Further Work

As previously mentioned, the 2008 NSF workshop to define Computational Thinking was unable reach a strong consensus. Still, many notable arguments were made during the process. In order to strengthen and expand the Unified Model of Computational Thinking, these arguments should be re-examined in light of this new definition. This can be used to identify weaknesses in the model . This approach can be applied similarly to all research on Computational Thinking, with perhaps a more restrictive mindset (given the highly variable quality of research).

Figure 2. The General Unified Model of Computational Thinking

Hard Skills	
Data Abstraction	Identifying important elements and filtering out unimportant ones, in the context of a problem, especially in a way amenable to being represented by a computer or other tool.
Data Organization	Logically organizing data, especially at varying scales of complexity.
Data Analysis	Finding patterns and connections in data, especially using a computer.
Problem Formulation	Using formal, precise notation to define a problem, especially generalizing the problem into one with a known solution.
Problem Decomposition	Decomposing the problem into manageable pieces, especially using a strategy such as Divide and Conquer or an Iterative approach.
Problem Analysis	Analyzing a given problem in terms of computability, with a focus on whether the computer is a suitable tool for solving this problem.
Algorithm Design	Using formal, precise notation to define the steps of a solution, especially in such a way that the steps can be executed by a computer.
Algorithm Generalization	Recognizing when a known solution can be applied to a new problem.
Algorithm Analysis	Comprehending and evaluating an algorithm, especially in regards to how it performs in certain aspects (e.g., time, space, energy, exertion).
Soft Skills	
Creativity	Creating new data, problems, and algorithms in order to satisfy a humanistic imperative, especially using Hard computational skills.
Communication	Conveying problems, data, and algorithms to peers, computational experts, domain experts, and the wider public, both in the short term (e.g., describing a diagram to someone in the same room as you) and long term (e.g., creating a blog post).
Collaboration	Working with peers, computational experts, domain experts, and the wider public to identify new problems, data, and algorithms. Notice that this is distinct from Communication, which is a unidirectional flow of information, and Creativity, which is a solitary action; Collaboration combines these two concepts as a new paradigm in order to stress the social aspect of Computational Thinking.
Attitudes	
Confidence with complexity	Recognizing that problems, data, and solutions come in many sizes, and that Computational Thinking gives us the leverage to work with even the biggest problems.
Persistence in Difficulty	Having patience when dealing with challenging problems that might push the way you think, and not allowing frustration to affect your evaluations.
Tolerance for ambiguity	Avoiding a negative emotional response to incomplete data, problems, or algorithms.

** Algorithms is used here in the most general sense as a ”series of steps”. Programming is an example of Algorithm Design, but it is not the entire idea.*

4 Proposed Pedagogy

In this section, we propose a dramatic change in the undergraduate program to include stronger elements of Computational Thinking. We look at the goals of such a program, and then discuss how a modular design can help achieve said goals.

4.1 Educational Goals

When designing a radical program change such as this, it is important to identify what the goals of the new program are. Obviously, the operational objective is to create new Computational Thinkers, students who have mastery of the skills and carry the attitudes detailed in the General Unified Model. However, there are a number of underlying goals for this objective:

Improve students performance in their core discipline Computational Thinking offers new techniques and approaches to using technology; it has already revolutionized Biology (re: Computational Biology) and a number of other fields. There are strong arguments that further innovations can be expected simply by preparing students with the right knowledge. [24]

Improve interdisciplinary education and research The idea of a ”liberal college education” includes the idea of breadth - that students should be exposed to a wealth of knowledge outside of their core discipline. Computational Thinking, by virtue of its abstract nature, is a natural lead-in to analyzing other domains. [24]

Bring more students to STEM and, in particular, to Computer Science. Forecasts indicate that by 2020, there will be less than a third of the requisite 1.4 million programmers needed to fill computing-related jobs. [10]. By creating Computational Thinkers, we can help close this gap with computationally-enabled domain experts.

Transform attitudes about Computer Science The Computer Science community has a publicity problem, and efforts in the past decade have struggled to move past it. The limited and sometimes negative public perception of Computer Science has seriously affected enrollment [4]. By providing students new insight into what Computer Science is, a ripple effect can occur.

Educate students on the power of Computers Computers are an integral part of society, and most students interact with them constantly. This trend will only grow in the future, so it behooves educators to prepare students as much as possible to know how to interact with these devices.

Analyzing these goals, a tension emerges: we simultaneously want to draw in new Computer Science majors while strengthening other students in their core discipline, all while creating new interdisciplinary workers that can comfortably shift between these two poles. To support these two directions, we need a flexible curriculum that supports transition and mediates transfer.

4.2 The MUSIC Model of Academic Motivation

Humanistic learning theories stress the importance of recognizing the human role in learning. Education is not just a simple accumulation of facts, or even the development of skills; it is something that humans pursue with intentionality and values. Therefore, when developing a curriculum, it is important to recognize the effect of ”self, motivation, and goal-setting” [13].

Although there are many theories and frameworks available to study and promote motivation in students, each with different emphases and focuses, this paper will use the MUSIC Model of Academic Motivation by Jones. This model is particularly suited to Instructional Design and has strong validation [17]. This model can be employed at multiple levels of instruction, from the program down to individual assignments. It is composed of five elements [16]:

eMpowerment Do students believe that they have control over some aspects of their learning?

Usefulness Do students understand why what they are learning is useful to their interests, to their career goals, and/or in the real-world?

Success Do students believe that they can succeed if they put forth the effort?

Interest Do students demonstrate an interest in the course activities?

Caring Do students believe that you care about whether they achieve the course objectives?

4.3 Proposed Program

Our proposed program has two parts: an introductory course for freshman and a junior-level seminar course. The former lays groundwork that should recur throughout the students undergraduate career, and the latter acts as guide for measuring and building long-term transfer of that groundwork.

4.3.1 First-level course

Eventually, every freshman in the university will be required to take a Computational Thinking course either their first or second semester, in much the same way that every student is currently required to take an English course (with the logical exception of english majors, which corresponds to Computer Science majors being exempt from the CT course). In order to optimially design these courses, collaborations will occur between Computational Thinking experts, Instructional Design experts, and Subject-matter experts. Unfortunately, this aspect of the project will be extremely ad-hoc. Therefore, a design-based research methodology will be used. First and foremost, the growth of the CT program will occur in phases; the first semester of this project, for instance, will only involve 3-5 departments. Formative post-analysis of the successes and mistakes of that endeavor will feed the second iteration, which will involve a larger percentage of the undergraduate majors.

Naturally, the Computational Thinking course found in the Biology department will be wildly different from the one found in the Criminal Justice department. Taken by themselves, these courses would lead to a narrow and limited view to what Computational Thinking is. Furthermore, many students will be insufficiently aware of their long-term goals to choose an adequate major. Therefore, the courses will be divided into three units, each punctuated by assessment, preceded as a whole by a generalized introduction, and ending with a massive symposium attended by all undergraduates enrolled in all of the parallel computational thinking courses. Specifically:

A) Generalized Introduction The first classes of every CT course will begin with a definition and explanation of generalized Computational Thinking, drawing on not only the Unified Model, but other competiting definitions from the literature. It will speak of the benefits of Computational Thinking and the value that this course is expected to offer all students. In fact, this is directly meant to build student Interest and sense of Usefullness, in accordance with the MUSIC model. Additionally, offering a consistent introduction across the student base should ideally synchronize student's understanding of the term.

B) Unit 1: Computational (Major) The first unit will contain material related to Computational Thinking's application within the department's discipline. The exact content will vary wildly between departments, but should be based on the Unified Model as strongly as possible.

C) Assessment 1 Assessment measures will be discussed more closely in the next section of this paper; for now, it is sufficient to describe the assessment instrument as measuring two components: abstract understanding of Computational Thinking and contextualized understanding of Computational Thinking. The most important aspect of this assessment is the intent to offer students feedback, and guide their understanding of Computational Thinking.

D) Unit 2: Computational (Tangent Major) The second unit will contain material related to another, tangentially related field. There is flexibility in deciding what this tangent major should be. It is the author's recommendation that the major be selected by the class itself, in order to directly appeal to student's sense of eMpowerment. However, it is reasonable to expect departmental regulations in this regard (perhaps the Animal Science department sees this an excellent opportunity to ensure that students get a healthy dose of Computational Biology), or teacher's perogative (if a Criminal Justice professor is knowledgeable about the subject, they might decide to teach Computational Psychology).

E) Assessment 2 This secondary assement will be similar to the first, once again with the intent of offering feedback on a students understanding of Computational Thinking.

F) Unit 3: Open-ended Project The third unit will be an open-ended project-based learning experience where students can explore Computational Thinking topics that are of interest to them. This freedom will be tempered by a common set of expectations for students: their project must have a significant Computational Thinking element, and should either explore how a problem can be solved using Computational Thinking, Analyze an existing solution using Computational Thinking, or result in the creation of an artifact using Computational Thinking. Additionally, the scope of the project should be appropriate to the length of time available to them (most likely 3-4 weeks). Finally, students will be strongly encouraged to work in teams of 2-3 to develop Collaborative skills.

G) Computational Thinking Symposium In the week before the end of the semester, there will be one or more symposiums showcasing students' work. Every team or individual will demonstrate or report their work. Additionally, students will be expected to evaluate two other teams work in light of the principles of Computational Thinking. The expectation is that students will gain an even greater understanding of the breadth of contexts that Computational Thinking applies to.

H) Assessment 3 The final assessment will be in the style of the first two, but will be focused less on feedback and more on evaluation.

Final course grades will be a mixture of individual assignment grades from Units 1 and 2, participation credit in assessments 1 and 2, peer and instructor evaluations of the final project, and the analysis of the final assessment. Exact weights for each of these elements will be determined by the professor and department, although ideally some general consensus should be reached.

In order to handle the university-wide scope of this endeavor, course-agnostic ”Computational Thinking” TAs will be provided by the Computer Science department to supplement classrooms. These TAs will be responsible for assisting with students projects, guiding when they are able, and forwarding them to useful resources, such as those provided by Google[1] and the ISTA[2], when they can't. Although these TAs will initially come from the Computer Science department, eventually the programs will bootstrap their own TAs as students complete the Computational Thinking course.

Each department will be responsible for working with Computational Thinking and Instructional Design experts to develop their unit's curriculum. However, the materials, lectures, and resources they develop will be open-source, and heavy-sharing is expected. Fortunately, a large number of resources exist already in the literature to serve as a base. It is important to note that there will be much diversity between approaches to teaching the units; some disciplines may find that programming is unnecessary to teach these skills, or that a visual programming environment is sufficient, or that students need a full programming language. It will be up to the collaborative development team to determine what is feasible in the time allocated for the semester.

4.3.2 Followup Workshop

In an ideal world, students would encounter Computational Thinking throughout their entire undergraduate program. In reality, to have these experiences we must make an additional change to the programs. Therefore, in student's junior year, 1-credit seminar courses will be held that brings together students of many diverse majors. These student groups will meet once a week for an hour to discuss how Computational Thinking can be applied to their majors. Each student will be responsible for writing a short essay on this subject, which will be presented to their peers. This is an excellent opportunity for students to develop their Communicative skills; they will be evaluated more strongly on their ability to communicate the problems and solutiosn than they will on the content. In addition to this subjective assessment, students will be assessed one final time, using the general Computational Thinking assessment instrument component from freshman year.

5 Assessment

Assessment is a complicated issue, but a critical component in education research projects. At the micro-level, we need to measure how much Computational Thinking has been ingrained in students, and at the macro-level we need to identify how successful our program changes have been. The former will be tied to our General Unified Model, and the latter will be tied to our educational goals. Still, neither will be easy to measure.

[1]`www.google.com/edu/computational-thinking/`
[2]`http://www.iste.org/learn/computational-thinking`

5.1 Student Understanding

First of all, what does it mean to understand Computational Thinking? Our definition offers a starting point, but it is meant to be extended. As such, leeway should be given as domain experts collaborate with the CT experts to design their assessment. However, it is critical that they settle on an operational definition of what is to be learned. These definitions will vary heavily between disciplines, although they will all bear similarity to the main themes of Problems-Data-Algorithms that was present in the General Unified Model, e.g., being able to frame, abstract, and solve a general form of biology problem in computational terms.

Since we want students to be able to do more than recite a definition; we want them to have a changed way of thinking. This implies that CT is more of a skill than a series of facts, which means that it should be approached from a situated learning perspective. There are a number of frameworks and tools available to assess this kind of learning. For example, [11] gives a five-dimensional framework for designing authentic assessment, building strongly off situated practices by asking designers to think not just about the task and its results, but also the social and physical context. Other useful tools [6] include:

Portfolios Over the course of a semester, students should inevitably create a number of artifacts foor their portfolio – the process of reviewing these creations at the end of a unit or semester and selecting the best ones is a useful pedaogogical activity. They are also useful for evaluating student's growth over the course of the semester and their evolving understanding and skill.

Performance Assessments In order to see if students can perform a skill, a simple measure is to ask them to perform that skill in front of the instructor. If students were learning programming, then this is an opportunity for them to be given a domain problem and then to code a solution; if they didn't learn any programming, then they might describe the solution in a high-level conceptual way. Many variations on this theme are possible. For instance, students might be given an existing algorithm and asked to debug it [23], asked to analyze and improve the organization and abstraction of some data, or to analyze scenarios for potential problems that could be solved computationally [18].

Concept Maps The weakness of traditional recall-based asseessment, such as true/false and multiple-choice, is that they are susceptible to memorization tricks rather than deep understanding. Requiring students to build concept maps, where they link related ideas and definitions, can reveal a more contextualized understanding indicative of mastery.

In addition to measuring hard skills, we must also measure soft skills and attitudes. These are usually measured subjectively; students working in teams can review partners, for instance [3]. These can be supplemented with the observations of the professor and any physical logs that are present, such as commit logs.

Of course, the design of our instructional units means that great autonomy has been given to each department to design their curriculum, and assessment methods should ultimately be tailored to their creations. Some might find that having students write descriptive essays of what Computational Thinking means, or taking multiple-choice quizzes on domain-related CT tools, or that any other more traditional form of assessment is appropriate for their classroom. Just as with building instruction, to build assessment the domain experts and computational thinking experts need to work directly with instructional design experts.

The repeated assessments are meant to be more diagnostic than evaluative. Early feedback to the student about their understanding will help drive their understanding. Ultimately, grades must be given to students. Important student assessments occur at the end of Unit 1 and 2 (contextualized and general), the end of the course (general only), the end of the workshop (general only). Additionally, student end-of-course projects and workshop presentations will be assessed. Finally, grades can be influenced strongly by participation and attendance, as the instructors desire.

5.2 Program Success

We previously established a number of goals for this project, and in this section we suggest several ways that they can be evaluated. Important program assessements will occur at the end of the semester. the iterative, design-based research approach means that assessment measures may change over time.

Improve students performance in their core discipline Ideally, student performance will change as a result of the introduction of this new technique. There is low-to-no expectation that grades will change sharply simply from learning this new skill, but should be strong expectation that qualtiative analysis gathered during the workshop on the ways that students view their discipline will reveal that students opinions have changed.

Improve interdisciplinary education and research This is a conceptually simple to track, by looking at research projects being undertaken at the university as the cohorts taking these computational classes enter the research pipeline. However, the task of analyzing all the ongoing projects at a top-teir research university should not be underestimated; it is almost a research project unto itself.

Bring more students to STEM and, in particular, to Computer Science. Exit surveys of the Computational Thinking class should check whether students are furthering their Computer Science education, whether by switching majors or adding a minor. If there is sufficient interest, this would be a valuable time to offer a Computational Thinking Certification from the department. Long-term enrollment changes should definitely be tracked alongside transitioning students' performance in subsequent computer science classes.

Transform attitudes about Computer Science Unfortunately, attitudes are a subjective dimension, and can only be assessed by humans in a subjective manner. Fortunately, there are a wide range of attitude surveys available for Computer Science, as this is a heavily researched topic in Computer Science Education. By using these surveys pre- and post- the lecture the workshop, an accurate map of the change in students' attitudes should develop.

Educate students on the power of Computers Similar to the attitudes, the conceptual knowledge of the power of computer can be measured using quantiative surveys. However, it should also be analyzed qualitatively during the workshop when students describe how the course has impacted their day-to-day lives.

6 Conclusion

The power and utility of computers continues to grow, and it behooves Computer Science to share the benefits with other disciplines. In this paper, we describe how a university could alter its undergraduate program to define, teach, and assess one of the key essences of Computer Science – Computational Thinking. This change will affect all disciplines in the university and is expected to have wide-reaching impacts. Of course, interdisciplinary work is a two-way street; the possibility should be left open that by sharing a little bit of Computer Science with the world, the world might change Computer Science.

A Weinberg Computational Thinking Research Taxonomy

The following taxonomy is taken from [**?**] and classifies the various types of papers in the literature.

Curriculum Description (48%) Articles that explain a lesson, curriculum, activity, or course that is used to promote computational thinking or one of its domains

Program Description (7%) An intervention or idea that goes beyond the individual classroom level and is implemented on a larger scale across a university, for example.

Evaluations (5%) The primary aim for the paper was to convey the results of a study focused on a specific program or intervention. This goes beyond the prior two sections in that they emphasize evaluation methods and findings.

Research (12%) Distinguished from evaluations by their focus on informing theory or contributing to the larger knowledge base rather than being focused on a single program or intervention.

Philosophy (23%) Papers intended to create or promote debate about computational thinking in the broad computer science or computer science education communities. Works in this category might aim to share a perspective on an issue within the field or describe how CT applies to other disciplines.

Opinion (6%) Similar to Philosophy, except with a more subjective perspective; they are typically not peer-reviewed.

B Dominant Definitions of Computational Thinking

B.1 AP CS Principles

P1: Connecting computing

Developments in computing have far-reaching effects on society and have led to significant innovations. These developments have implications for individuals, society, commercial markets, and innovation. Students in this course study these effects and connections, and they learn to draw connections between different computing concepts. Students are expected to:

- Identify impacts of computing;
- Describe connections between people and computing; and
- Explain connections between computing concepts.

P2: Developing computational artifacts

Computing is a creative discipline in which the creation takes many forms, ranging from remixing digital music and generating animations to developing websites, writing programs, and more. Students in this course engage in the creative aspects of computing by designing and developing interesting computational artifacts, as well as by applying computing techniques to creatively solve problems. Students are expected to:

- Create an artifact with a practical, personal, or societal intent;
- Select appropriate techniques to develop a computational artifact; and
- Use appropriate algorithmic and information-management principles.

P3: Abstracting

Computational thinking requires understanding and applying abstraction at multiple levels ranging from privacy in social networking applications, to logic gates and bits, to the human genome project, and more. Students in this course use abstraction to develop models and simulations of natural and artificial phenomena, use them to make predictions about the world, and analyze their efficacy and validity. Students are expected to:

- Explain how data, information, or knowledge are represented for computational use;
- Explain how abstractions are used in computation or modeling;
- Identify abstractions; and
- Describe modeling in a computational context.

P4: Analyzing problems and artifacts

The results and artifacts of computation, and the computational techniques and strategies that generate them, can be understood both intrinsically for what they are as well as for what they produce. They can also be analyzed and evaluated by applying aesthetic, mathematical, pragmatic, and other criteria. Students in this course design and produce solutions, models, and artifacts, and they evaluate and analyze their own computational work as well as the computational work that others have produced. Students are expected to:

- Evaluate a proposed solution to a problem;
- Locate and correct errors;
- Explain how an artifact functions; and
- Justify appropriateness and correctness.

P5: Communicating

Students in this course describe computation and the impact of technology and computation, explain and justify the design and appropriateness of their computational choices, and analyze and describe both computational artifacts and the results or behaviors of such artifacts. Communication includes written and oral descriptions supported by graphs, visualizations, and computational analysis. Students are expected to:

- Explain the meaning of a result in context;

- Describe computation with accurate and precise language, notation, or visualizations; and
- Summarize the purpose of a computational artifact.

P6: Collaborating

Innovation can occur when people work together or independently. People working collaboratively can often achieve more than individuals working alone. Students in this course collaborate in a number of activities, including investigation of questions using data sets and in the production of computational artifacts. Students are expected to:

- Collaborate with another student in solving a computational problem;
- Collaborate with another student in producing an artifact; and
- Collaborate at a large scale

B.2 CSTA

Computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

1. Formulating problems in a way that enables us to use a computer and other tools to help solve them.
2. Logically organizing and analyzing data
3. Representing data through abstractions such as models and simulations
4. Automating solutions through algorithmic thinking (a series of ordered steps)
5. Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
6. Generalizing and transferring this problem solving process to a wide variety of problems

These skills are supported and enhanced by a number of dispositions or attitudes that are essential dimensions of CT. These dispositions or attitudes include:

1. Confidence in dealing with complexity
2. Persistence in working with difficult problems
3. Tolerance for ambiguity
4. The ability to deal with open ended problems
5. The ability to communicate and work with others to achieve a common goal or solution

B.3 Google Definition

Computational thinking (CT) involves a set of problem-solving skills and techniques that software engineers use to write programs that underlie the computer applications you use such as search, email, and maps. Here are specific techniques: [14]

Decomposition The ability to break down a task into minute details so that we can clearly explain a process to another person or to a computer, or even to just write notes for ourselves. Decomposing a problem frequently leads to pattern recognition and generalization, and thus the ability to design an algorithm.

Pattern Recognition The ability to notice similarities or common differences that will help us make predictions or lead us to shortcuts. Pattern recognition is frequently the basis for solving problems and designing algorithms.

Pattern Generalization and Abstraction The ability to filter out information that is not necessary to solve a certain type of problem and generalize the information that is necessary. Pattern generalization and abstraction allows us to represent an idea or a process in general terms (e.g., variables) so that we can use it to solve other problems that are similar in nature.

Algorithm Design The ability to develop a step-by-step strategy for solving a problem. Algorithm design is often based on the decomposition of a problem and the identification of patterns that help to solve the problem. In computer science as well as in mathematics, algorithms are often written abstractly, utilizing variables in place of specific numbers.

B.4 Cuny, Snyder, Wing

Computational thinking for everyone means being able to

- Understand what aspects of a problem are amenable to computation
- Evaluate the match between computational tools and techniques and a problem
- Understand the limitations and power of computational tools and techniques
- Apply or adapt a computational tool or technique to a new use
- Recognize an opportunity to use computation in a new way
- Apply computational strategies such divide and conquer in any domain

Computational thinking for scientists, engineers, and other professionals further means being able to

- Apply new computational methods to their problems
- Reformulate problems to be amenable to computational strategies
- Discover new science through analysis of large data
- Ask new questions that were not thought of or dared to ask because of scale, easily addressed computationally
- Explain problems and solutions in computational terms

References

[1] T. C. Board. The national science foundation provides $5.2 million grant to create new advanced placement computer science course and exam, 2013.

[2] A. Bundy. Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1(2):67–69, 2007.

[3] R. Burns, L. Pollock, and T. Harvey. Integrating hard and soft skills: Software engineers serving middle school teachers. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 209–214. ACM, 2012.

[4] L. Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, SIGCSE '06, pages 27–31, New York, NY, USA, 2006. ACM.

[5] L. N. Cassel. Interdisciplinary computing is the answer: Now, what was the question? *ACM Inroads*, 2(1):4.

[6] J.-I. Choi and M. Hannafin. Situated cognition and learning environments: Roles, structures, and implications for design. *Educational Technology Research and Development*, 43(2):53–69, 1995.

[7] C. Day. Computational thinking is becoming one of the three R's. *Computing in Science and Engineering*, 13(1):88–88, 2011.

[8] P. J. Denning. The profession of IT: Beyond computational thinking. *Communications of the ACM*, 52(6):28–30, 2009.

[9] I. S. for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA). Operational definition of computational thinking, 2011.

[10] N. C. for Women and I. T. (NCWIT). Ncwit's women in it: By the numbers, 2013.

[11] J. Gulikers, T. Bastiaens, and P. Kirschner. A five-dimensional framework for authentic assessment. *Educational Technology Research and Development*, 52(3):67–86, 2004.

[12] D. Hemmendinger. A plea for modesty. *ACM Inroads*, 1(2):4–7, 2010.

[13] W. Huitt. Humanism and open education, 2009.

[14] G. Inc. Exploring computational thinking, 2011.

[15] J. A. Jacobs. Interdisciplinary hype. 2009.

[16] B. D. Jones. Motivating students to engage in learning: The MUSIC model of academic motivation. *International Journal of Teaching and Learning in Higher Education*, 21(2):272–285, 2009.

[17] B. D. Jones and G. Skaggs. *Validation of the MUSIC Model of Academic Motivation Inventory: A measure of students' motivation in college courses*. Research presented at the International Conference on Motivation 2012, 2012.

[18] K. S. Marshall. Was that ct? assessing computational thinking patterns through video-based prompts. *Online Submission*, 2011.

[19] S. Papert. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, 1(1):95–123, 1996.

[20] J. Pfeiffer. *The thinking machine*. Philadephia, PA: Lippincott, 1962.

[21] J. Randolph, G. J., E. Sutinen, and S. Lehman. A methodological review of computer science education research. *Journal of Information Technology Education*, 7:135–162, 2008.

[22] A. Weinberg. Computational thinking: An investigation of the existing scholarship and research. In *School of Education*. Fort Collins, Colorado State University, 2013.

[23] L. Werner, J. Denner, S. Campe, and D. C. Kawamoto. The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 215–220. ACM, 2012.

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