ABSTRACT
As computer science education (CS Ed) research matures, CS Ed researchers need to take a step back and consider more foundational questions of what it means to know how to program. Along this journey we will need to use research frameworks and methodologies from learning research that help us better understand who our students are as learners and how that affect how they learn, their motivation for learning, and how best to support their learning. In particular, we need to gain a better understanding of the mental modes students build around important programming constructs and concepts and the meta-cognitive skill necessary for learning to program and consistent application of programming and computational thinking skills. Lastly, we need to develop an understanding of the differences between novice and experts to begin defining a computer science specific pedagogy that distinguishes between content knowledge we want students to know and the pedagogical content knowledge needed to support students’ learning that content. While a few researchers in the field have already begun to branch into these research directions, this research has not matured enough that everyone recognizes these works as foundational to all the other computer science education research we conduct.

Keywords
Learning, Meta-cognition, pedagogical content knowledge, performance and learning goals, mental models, task analysis, and misconceptions

1. INTRODUCTION
For many years now computer science education research has focused on figuring out what content students need to be taught, debating about what students’ first programming environment should be, designing visual programming languages, and redesigning CS1. However, learning is more than content knowledge and tools to support programming. Learning is a complex process that requires students to actively engage in learning new content, accessing previously learned content, reflection on what they know, don’t know, and do not understand yet. In the process, learners must construct and reconstruct mental models of content and problem solving strategies, and understand context of use of these models to solve problems. Learning to program is a complex task that requires students to monitor their understanding about a concept, identify the correct programming construct to develop a solution, and recognize when both their thinking and their problem solving strategies are working or not working. In this paper, I argue that by using research frameworks and methodologies from learning research there are several viable avenues of research that should be undertaken by computer science education researchers to ensure that our novel interventions: programming environments, camps, and curricular units, have their desired long-term impact. These research frameworks and methods will help us to answer the following research questions:

What does it mean for someone to know how to program?

- How do students learn to program and what does that development look like?
- What is the cognitive load of students who are learning to program and programming?
- What are successful and unsuccessful mental models of challenging programming concepts?
- What are common challenges in conceptual understanding in computing courses?
- What meta-cognitive skills and strategies are needed to learn to program and apply programming knowledge and novel situations?
- What are common transfer issues across the undergraduate computing curriculum?
- How do we support transfer of programming skills from starter programming environments like Scratch, Alice, MIT App Inventor, Greenfoot etc, to text-based programming languages and IDEs?

2. TOWARD UNDERSTANDING HOW PEOPLE LEARN TO PROGRAM
Thus far, computer science education research has focused on helping students to learn foundational computing concepts such as loops, iteration, conditionals, variables, statements, and data structures, and then turns to much higher level computing concepts such as operating systems, computing theory, software design and testing, and other specialty areas. However, what is missing is an understanding of the underlying mental models that students hold with respect to these foundational concepts, the strategies needed to apply their knowledge and design skills and underlying principles they need to organize their thoughts in predefined and open-ended problem spaces such as bringing a project from idea to implementation. In this section we will explore the need for task analyses and mental models of programming tasks and environments; greater understanding of meta-cognition needed to successfully program, understanding the
differences between experts and novices and our preconceived notions of what makes a good programmer, and issues of transfer.

2.1 Mental Models and Task Analyses
This work is among the most important work that computer science education researchers need to conduct. This work begins by conducting tedious analyses of programming tasks, IDE and tool use, and courses to better understand what is required by students to solve problems. In addition, parallel research into the processes and strategies students actually use to learn to program, apply programming knowledge to solve problems. Several researchers have already begun doing this research but more work needs to be done to validate these results across the field [8, 11, 12]. The benefits of this work entails identification of the mismatch between the processes and strategies we expect students to engage in and the one’s they actually employ. In addition, this research will reveal the common misconceptions by both students and teachers. Overall, exploring students’ mental models and conducting task analyses allows us to concurrently take a tops-down and a bottoms-up approach to understanding how people learn and the cognitive and physical activity required to learn to program.

2.1.1 Task Analyses
Task analysis is a top-down approach to understanding what is required for students to learn to program and become computer scientists. Task analysis is the analysis of how a task is accomplished. Task analyses commonly include detailed descriptions of manual and mental/cognitive activities, task duration, frequency, and complexity, conditions under which tasks should be or are undertaken, and other factors involved in or required for a person or group of people to perform a task [9]. Task Analyses are used in a number of fields, Human Computer Interaction, education, health care, business to better design systems, interfaces, training, and support and operating processes.

In computer science, we would greatly benefit from conducting task analysis on multiple levels: assignments, IDEs and programming environments, and courses.

Task analyses on programming assignments and projects, would help us to better understand the prior knowledge we are assuming students should have, the ways in which they should be using their knowledge, the cognitive resources required to learn to program and to use that programming ability. These analyses will reveal the complexity of the tasks we are assigning and the mental and cognitive activity needed to complete these assignments.

Task analyses on IDEs and programming environments will reveal the multiple aspects of the environment that students must cognitively attend to as well as physically manipulate in order to design, organize, compile, and run their code. In conjunction with the analyses of the assignment and the tools needed to design the solution, we will be surprised about the level of cognitive engagement we are expecting students to engage in even on an introductory level.

If we conducted a task analysis on all the courses in our curriculum it would help us to identify the parts that students really struggle with in terms of content, process, and cognitive requirements. It would help use to outline the prior knowledge dependencies between courses, gaps in our instruction with respect to processes and strategies students need to learn but we assume they already have.

Overall, computer science education research would benefit from task analysis in order to define the complexity and cognitive demand of our instruction and instructional practices and put us on the road toward developing appropriate scaffolds to support struggling learners and to increase productive learning.

2.1.2 Mental Models, Schema, and Knowledge Representation
While task analyses will provide the field with a better picture of the physical and mental demand of our assignment, curricula, and tools from the top-down, eliciting students and teachers’ mental models, schema, and knowledge representations will help us to understand how learners understand and use content, common misconceptions, and mistakes they make, and insight into how to solve these issues from the bottom-up.

Mental Models and Schema are particular kinds of knowledge representations. As students are learning, constructivist theories of learning suggest that they are actively constructing and re-constructing their knowledge and organization of this knowledge [7]. Mental Models are internal representations that people form about the world that included concepts, objects, and systems. People use these mental models to organize information and to run mental simulations to predict the behavior of objects, concepts, and systems they encounter. Schema are similar to mental models but their focus is more on organization of information to help people identify important features to attend to, exception, exemplars, etc.

Eliciting mental models and schemas can be difficult because people aren’t often aware of the exact structure of their mental representations. However, by having students and teachers participate in cognitive walk-through and think-alouds, we can begin to understand the differences between teachers and students’ representations of programming tasks, assignments, and tools. We can begin to better understand common misconceptions students have with respect to programming constructs, why some concepts are more difficult to understand than others, and why some misconceptions are so difficult to break.

Mental models, schema, and other knowledge representations are discipline specific constructs. Thus, solid research needs to be conducted on the types of mental models that students and teachers form around important computing concepts.

In order to conduct task analysis and mental model research, laboratory studies needed to be designed for K-12 and undergraduate students and teachers. If funded projects like these can significant results within five years if not earlier due to the focused nature of the study and analysis.

Once this knowledge is obtained we can move onto in-situ research studies that evaluate the validity of these measures in classroom settings where nothing quite goes as you plan.

2.2 Meta-cognition
In addition, to our lack of understanding of the what it takes mentally for students to understand concepts we lack an understanding of the underlying principles that students need to organize their thoughts in an open-ended problem space such as bringing a project of their own devising from idea to implementation and the related strategies needed to apply their knowledge and design skills.

In order for students to effectively learn computing concepts and to design sophisticated programs, students need to engage in
meta-cognition. Meta-cognition is the thinking a person does about their own thinking. Meta-cognition allows students to engage in self-directed learning and other-directed learning. This is particularly important in computer science because so much of what students are exposed to in a computer science class is what they’ve seen for the first time. Thus, by researching how students reason about content they have not been previously pre-disposed to attend to, such as variables, redundancy, and common problem solving errors, we will gain a better understanding of how students can learn computer science through effort and will. In addition, learning to program requires a very high level meta-cognition. Meta-cognition develops gradually and is dependent on knowledge as experience. The big take way for computer science education researchers is that it is difficult to engage in self-regulation and reflection in areas that one does not understand.

We can adapt micro-genetic studies from psychology and cognitive science to study strategy development of computer science students. Micro-genetic studies in other research areas have revealed that new strategies are recognized in the context of successful performance and not in response to impasses or failure; short-lived transition strategies often precede more enduring approaches; and new approaches often occur very slowly [3]. This is significant for computer science education research because more often than not we allow students to fail with the hope that they will learn from their mistakes. This research suggests that our students might do better to have more success and less failure on their road to becoming computer scientist and computer science educators.

Moreover, coming to an understanding of how people develop programming strategies can lead to instructional that focus on helping students understand how strategies can help them solve problems, to recognize when each strategy is likely to be most useful, and to transfer strategies to novel situations [3].

2.3 Experts vs. Novices

“Research shows that it is not simply general abilities, such as memory or intelligence, nor the use of general strategies that differentiate experts from novices. Instead, experts have acquired extensive knowledge that affects what they notice and how they organize, represent, and interpret information in their environment. This, in turn, affects their abilities to remember, reason, and solve problems [1:31].”

Research into the differences between experts and novices in computer science education is relevant because we make a lot of assumptions about what students know and don’t know and how to best teach them. However, without a strong knowledge base of understanding the difference between novice and expert knowledge, we could be setting unrealistic expectations for students and teachers. In addition, this may lead to the design of courses, assignments, and assessments that do not effectively teach content or measure student understanding because they are focused on assessing the knowledge as organized by the instructor.

Moreover, in computer science education we have several types of novices and experts. We have K-12 novices, undergraduate novices, and teacher novices from K-12 with varying disciplinary and professional backgrounds, as well as undergraduates and graduate CS students. In addition, we have end-user programmer novices that are programming in applied contexts with varying levels of training [5, 10]. Our experts come in all different forms and levels of maturity and prior knowledge as well. We have professionals with undergraduate and graduate degrees, academics, and K-12 teachers. It is easy to see that all experts and novices are not created equal.

Thus, as the field moves forward it will be important understand these unique populations, the knowledge representations these learners and teachers construct as well as the match between the complexity of the tasks we assign them and the prior knowledge we assume.

Research into the difference between novice and experts in computing will help us define more explicitly in our instruction the meaningful features and meaningful patterns of information that are not noticed by novices but are often noticed by experts. In addition, we can become more cognizant and explicit about how content knowledge should be organized in order to promote deep understanding and reflection. We can distill strategies to help students identify appropriate contexts for application of content.

In order, to gain this type of knowledge we can use think-a-loud method provided for a very careful analysis of the conditions of specialized learning and the kinds of conclusion one can draw from them [6].

3. Answers to Research Questions in 5 Years

Developing research that elicits these knowledge representations as well as well constructed task analyses can help us to better answer the following questions:

- What is the cognitive load of students who are learning to program and programming?
- What are successful and unsuccessful mental models of challenging programming concepts?
- How do students learn to program and what does that development look like?
- What are common challenges in conceptual understanding in computing courses?

Answers to these questions will better prepare the field to address:

- Struggling students who have formed problematic mental representations
- Gaps in our instruction with respect to processes and strategies students need to learn but we assume they already have.
- Adjust the complexity of tasks and assignments and the cognitive load and demand on students to appropriate levels for the targeted content of each course for each population
- Design tools that reduce the already high cognitive demand of students learning to program or use programming expertise
- Design appropriate scaffolds to support students learning to better prepare them for the next course or for application of content in the real-world.

4. BENEFITS OF THESE NEW RESAERCH DIRECTIONS

Research like this has been conducted for decades in fields like psychology and education and more recently in the fields of cognitive and learning sciences. Thus, there are existing tools and methods for conducting this research efficiently and meaningfully.
This research will strengthen the research that has been conducted to date with empirical support as to why certain interventions have been so successful while others have not. In addition, it will strengthen the design of new programming IDEs, curricular interventions, and help us better understand how to support diverse sets of learners.

In particular, this research will yield a greater understanding of how students actually learn computer science and what is easy or hard. It will help the community understand how to better foster CS learning across the grade levels for all students and teachers. It will also help with the identification of strategies for overcoming common challenges by helping us understand the true source of the challenge.

The United States of America would benefit from this research as it would decrease attrition in computing major and increase the success of women and underrepresented minority students in the field. This increase of recruitment and retention will yield higher numbers of American students with degrees in computing fields. This will decrease the reliance of computing companies on foreign computing talent. Some may argue that this will increase the national security as it reduces the number of visas that need to be issued in technical fields. In addition, it increases the American talent available to work on critical systems with sensitive security clearance requirements.

5. REFERENCES