Diagrammatic Question Framework: Studying Effectiveness in First-Year Computing Courses

Lauren Himbeault
lauren.himbeault@umanitoba.ca
University of Manitoba
Winnipeg, Manitoba, Canada

ABSTRACT
Programming courses and instructors provide a wide variety of materials to help students learn how a computer will execute code. However, students may not engage with these resources effectively, or at all. This paper presents an initial framework for designing formative assessments, coined Code Diagram Queries (CDQs). The goal of CDQs is to act as a forcing function to engage students with formative diagrammatic material that will challenge and correct their current mental models for the notional machine of the language they are learning. Optional resources are often disregarded unless there are formative activities that drive students to interact with the materials. CDQs are questions that include or reference diagrams and are designed to drive student interaction with diagrammatic notional machine representations in a low-risk formative environment. This developmental environment allows students to explore new computer science concepts and expand on their understanding of them. I hypothesize that CDQs will help increase student engagement by leveraging diagrammatic materials and through this, help students accurately visualize and comprehend code execution. I will measure engagement with class materials and student confidence levels following their interactions with CDQs. This framework is meant to aid in the effective design of CDQs so that a student’s comprehension of code execution may be understood more deeply and then corrected as needed. The focus of this research is to provide new insights into the importance of guiding students through the construction of their mental models.

1 PROBLEM AND MOTIVATION
1.1 Learning a Notional Machine (NM)
Novice programming students face high failure and drop-out rates for many reasons [8, 13, 18, 20, 21]. A subset of reasons for these rates include difficulty with materials [16, 24, 29], self-efficacy [7, 22, 31], and prior experience [9]. Aiding students when they are faced with these challenges is important and previous research suggests direct instruction is beneficial for students [2, 26]. Educators employ different types of scaffolding to guide students as they expand their knowledge and push the bounds of their zone of proximal development [23, 25]. Using the correct type of scaffolding is imperative to support student learning and in this way, we can help ensure a deeper understanding of the NM [8].

1.2 Visual Resources
Visual tools such as static images, drawings, or even animated media are frequently employed to support learning. Evidence suggests static diagrams help students understand coding concepts [27] and work on the value of diagrams illustrates how a well-constructed image benefits a student’s comprehension and problem-solving skills [19]. A drawback of these static visualizations is their lack of interactivity. When a student is solidifying their understanding of a concept, the ability to interact with and/or modify a visual is crucial. An example of an animated visualization is PythonTutor, which is a web-based program that aids students understanding the NM [12].

1.3 Code Diagram Queries
Code Diagram Queries (CDQs) are a formative tool designed to aid students in developing accurate mental models of programming language NMs. I hypothesize that CDQs will act as a forcing function to get students to interact with NM visualizations and think about the way the computer interprets and executes code. These questions are intended to help reveal NM misconceptions with respect to a student’s understanding. Using this framework, CDQs have the potential to enhance student comprehension of the Python NM. This framework is not limited to the Python language and can be extended for other coding languages (Figure 1). Engaging students with activities designed to probe their understanding of a NM allows us to assess their mental model accuracy. The importance of using formative assessments for students when learning new concepts is well documented. Giving students the opportunity to explore concepts and attempt to understand them in a low-risk formative environment can help build confidence in coding [5, 15]. I have designed CDQs to be a low stakes assessment for students so they can explore and experiment with coding concepts.

2 BACKGROUND AND RELATED WORK
As NMs are abstract representations of code, educators often employ tangible renderings to help make concepts more concrete [11, 27, 28]. The concept of concreteness fading is often used when teaching an NM because of code execution being inherently abstract. Specifically, code execution refers to how the user-written code runs on a physical computer. Since different coding languages interact with computer architecture differently, the way in which these NMs are modelled must also vary. A plethora of tools exist to handle these deviations.

Some educators have used a more physical representation when displaying an NM [17, 19]. The benefits of physical presentations are numerous, however, these representations are not easily portable and can be cumbersome and tedious to manipulate. Unlike static representations of code, software for code visualizations exist for a more portable way to view and visualize code [4, 12, 14]. These web-based resources can help a student ensure the model they are looking at accurately represents the code they see as opposed to attempting to recreate what they saw in class. While different code
This paper presents an initial framework created to guide the design of CDQs, questions that query student’s understanding of a diagrammatic representation of an NM (Figure 1). I developed this framework using diagrams from PythonTutor [12], however, any visualization/animation that correctly displays code execution given the coding language and computer architecture could be the subject matter for CDQs. These assessments are intended to help students engage with the CDQ diagram and build a more accurate understanding of how a computer executes code. This framework identifies the cognitive levels at which the foundation of learning to program happens. The concepts of visualizations, NMs, and formative assessments create the basis upon which the CDQ framework is built. Specifically, this framework defines a design space for formative questions that span three core introductory programming concepts, the cognitive skills from Bloom’s taxonomy: Understanding, Applying, and Analyzing [3], and a variety of question formats. The methodology of using Bloom’s taxonomy classrooms is well studied, however this framework is designed to focus on 3 layers of the taxonomy [1, 30]. In coding, Bloom’s taxonomy has been thought to be difficult to incorporate but by focusing on specific layers of the taxonomy, this framework utilizes the taxonomy in a way that allows students to build strong foundations [10].

The CDQ framework is formed through the 3 layers of Bloom’s taxonomy, the 3 programming concepts outlined above, and different styles of questions. Special attention to the style of questions used delivering CDQs is necessary as learning is a spectrum and not all students feel confident with the same question types [6, 32]. Examples of varying questions targeting these processes for the concepts specified are found in Table 2.

The goal of this framework is to aide in the effective design of CDQs which probe and correct a student’s understanding of code execution. This framework is meant to be a guide in the design of formative assessments that use diagrammatic representations of NMs (CDQs) and to ensure coverage of multiple cognitive processes and coverage across the most critical early programming concepts. Thus, aiding students in building a more accurate understanding of how a computer executes code in a given language. Specifically, my framework defines a design space for formative questions that span from core introductory programming concepts, 3 different cognitive skill levels from Bloom’s taxonomy, and a variety of question formats. Even educators who are experts in their subject, working with novice programmers can be complex and understanding problems through their eyes can be challenging [16, 21]. This framework serves as a stepping stone for educators who are looking for new teaching techniques.

### 3 APPROACH AND UNIQUENESS

The primary goal of CDQs is to support students in their understanding of how code interacts with computer architecture, therefore building stable and accurate mental models. Students struggle with concepts such as how computers store and process data as part of a program. Rather than relying on a student constructing an accurate mental model through independent exploration, CDQs are a tool with which students can engage and these questions act as a forcing function to challenge the accuracy of a student’s mental model and alter it accordingly. CDQs will compel them to think differently and reshape their mental models appropriately. This research presents an initial framework that allows diagrammatic questions that make use of preferred code, known as Code Diagram Queries (CDQs) to be implemented using visualization tools.

Present research being conducted is analyzing the differential effectiveness of these CDQs and their ability to help students construct accurate mental models of computer processes in a first year computing class. While all visualization tools have limitations, this framework is designed so that it is not coupled with a particular schematic. This allows for flexibility in using different resources for the visuals depending on the language being employed and the visual tools already in use. CDQs are designed to be more effective at aiding in understanding an NM over existing non-diagrammatic activities. I hypothesize the inclusion of these questions will positively impact the engagement level of students.