# 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 helps 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

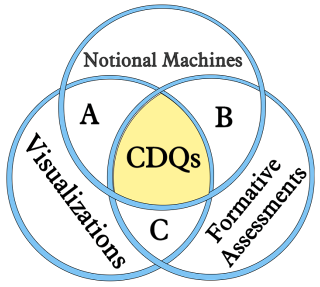
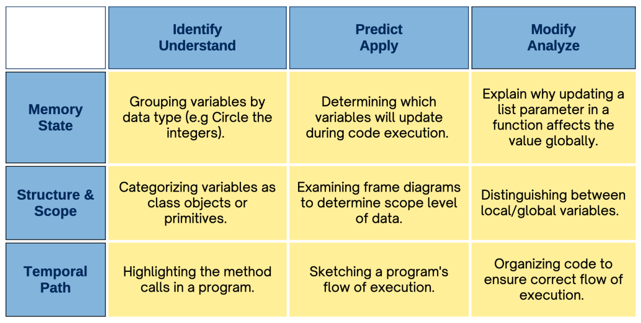


**Figure 1: The CDQ framework connects visualizations, NMs, and identifies the different cognitive levels at which formative assessments can take place. Intersections surrounding CDQs are shown: A. Visualization Software with no assessment component (Eg. PythonTutor); B. Formative assessments of an NM that do not involve visuals (Eg. multiple choice questions with no diagrams or code tracing tables); and C. Formative assessments using visualizations that do not explicitly address the NM (Eg. writing pseudo-code).**



| | Identify Understand | Predict Apply | Modify Analyze |
|---|---|---|---|
| Memory State | Grouping variables by data type (e.g Circle the integers). | Determining which variables will update during code execution. | Explain why updating a list parameter in a function affects the value globally. |
| Structure & Scope | Categorizing variables as class objects or primitives. | Examining frame diagrams to determine scope level of data. | Distinguishing between local/global variables. |
| Temporal Path | Highlighting the method calls in a program. | Sketching a program's flow of execution. | Organizing code to ensure correct flow of execution. |

**Figure 2: Examples of types of questions that can be asked accompanied with a diagrammatic representation of the code.**

visualizers exist, each with their pros and cons, many have common assets including the ability to move through animations at one's own pace and to experiment with your own code and not just use class examples or pre-selected algorithms.

## 3 APPROACH AND UNIQUENESS

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 code 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 3 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.

## 4 RESULTS AND CONTRIBUTIONS

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.

## REFERENCES

[1] Lorin W. Anderson, David R. Krathwohl, Benjamin Samuel Bloom, and Benjamin Samuel Bloom. 2001. . Longman, New York, NY, USA. 333 pages.

[2] Nicole Anderson and Tim Gegg-Harrison. 2013. Learning Computer Science in the "Comfort Zone of Proximal Development". In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 495–500. https://doi.org/10.1145/2445196.2445344

[3] Benjamin S" "Bloom. 1965. *"The taxonomy of educational objectives: Handbook 1"*. "Longman Higher Education", "Harlow, England".

[4] Michael Bodekaer Jensen & Mads Tvillinggaard Bonde. 2011. *Labster*. Labster. Retrieved Oct 2, 2022 from http://labster.com

[5] Carol Boston. 2002. The Concept of Formative Assessment. *Practical Assessment, Research, and Evaluation* 8, 9 (2002), 5. https://doi.org/10.7275/kmcq-dj31

[6] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 164–172. https://doi.org/10.1145/3105726.3106190

[7] E. Doyle, I. Stamouli, and M. Huggard. 2005. Computer anxiety, self-efficacy, computer experience: an investigation throughout a computer science degree. In *Proceedings Frontiers in Education 35th Annual Conference*. IEEE Proceedings Frontiers in Education 35th Annual Conference, Indianapolis, IN, USA, S2H–3. https://doi.org/10.1109/FIE.2005.1612246

[8] Benedict du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73. https://doi.org/10.2190/3lfx-9rrf-67t8-uvk9

[9] Nikita Dümmel, Bernhard Westfechtel, and Matthias Ehmann. 2018. Effects of a preliminary programming course on students' performance. In *Proceedings of the 3rd European Conference of Software Engineering Education*. ACM, New York, NY, USA, 77–86. https://doi.org/10.1145/3209087.3209088

[10] Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L. Lewis, Donna McGee Thompson, Charles Riedesel, and Errol Thompson. 2007. Developing a Computer Science-Specific Learning Taxonomy. *SIGCSE Bull.* 39, 4 (dec 2007), 152–170. https://doi.org/10.1145/1345375.1345438

[11] Emily R. Fyfe and Mitchell J. Nathan. 2019. Making "concreteness fading" more concrete as a theory of instruction for promoting transfer. *Educational Review* 71, 4 (7 2019), 403–422. https://doi.org/10.1080/00131911.2018.1424116

[12] Philip Guo. 2013. *PythonTutor*. PythonTutor. Retrieved Oct 2, 2022 from http://pythontutor.com

[13] Mark Guzdial. 2015. What's the best way to teach computer science to beginners? *Commun. ACM* 58, 2 (2015), 12–13. https://doi.org/10.1145/2714488

[14] Steven Halim. 2011. *VisualGo*. National University of Singapore (NUS) - Computing. Retrieved Oct 2, 2022 from http://visualgo.net/en

[15] John Hudesman, Sara Crosby, Bert Flugman, Sharlene Issac, Howard Everson, and Dorie B. Clay. 2012. Using formative assessment and metacognition to improve student achievement. https://eric.ed.gov/?id=EJ1067283

[16] Tony Jenkins. 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Vol. 4. Citeseer, LTSN Centre for Information and Computer Sciences, Loughborough, UK, 53–58.

[17] Colleen M. Lewis. 2021. Physical Java Memory Models: A Notional Machine. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 383–389. https://doi.org/10.1145/3408877.3432477

[18] Richard Mayer. 1981. The psychology of how Novices Learn Computer Programming. *Comput. Surveys* 13, 1 (1981), 121–141. https://doi.org/10.1145/356835.356841

[19] Syeda Fatema Mazumder, Celine Latulipe, and Manuel A. Pérez-Quiñones. 2020. Are Variable, Array and Object Diagrams in Java Textbooks Explanative?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 425–431. https://doi.org/10.1145/3341525.3387368

[20] Iain Milne and Glenn Rowe. 2002. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. *Education and Information Technologies* 7, 1 (2002), 55–66. https://doi.org/10.1023/a:1015362608943

[21] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming. *ACM Transactions on Computing Education* 18, 1 (2017), 1–24. https://doi.org/10.1145/3077618

[22] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. 2004. Self-efficacy and mental models in learning to program. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '04*. ACM Press, New York, New York, USA, 171. https://doi.org/10.1145/1007996.1008042

[23] Zak Risha, Jordan Barria-Pineda, Kamil Akhuseyinoglu, and Peter Brusilovsky. 2021. Stepwise Help and Scaffolding for Java Code Tracing Problems With an Interactive Trace Table. In *21st Koli Calling International Conference on Computing Education Research*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/3488042.3490508

[24] Adrian Salguero, William G. Griswold, Christine Alvarado, and Leo Porter. 2021. Understanding Sources of Student Struggle in Early Computer Science Courses. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA) *(ICER 2021)*. Association for Computing Machinery, New York, NY, USA, 319–333. https://doi.org/10.1145/3446871.3469755

[25] Karim Shabani, Mohamad Khatib, and Saman Ebadi. 2010. Vygotsky's zone of Proximal Development: Instructional Implications and teachers' professional development. *English Language Teaching* 3, 4 (2010), 237–248. https://doi.org/10.5539/elt.v3n4p237

[26] Jean Stockard, Timothy W. Wood, Cristy Coughlin, and Caitlin Rasplica Khoury. 2018. The Effectiveness of Direct Instruction Curricula: A Meta-Analysis of a Half Century of Research. *Review of Educational Research* 88, 4 (Jan. 2018), 479–507. https://doi.org/10.3102/0034654317751919

[27] Sangho Suh. 2019. Using Concreteness Fading to Model & Design Learning Process. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 353–354. https://doi.org/10.1145/3291279.3339445

[28] Anthony Trory, Kate Howland, and Judith Good. 2018. Designing for Concreteness Fading in Primary Computing. In *Proceedings of the 17th ACM Conference on Interaction Design and Children* (Trondheim, Norway) *(IDC '18)*. Association for Computing Machinery, New York, NY, USA, 278–288. https://doi.org/10.1145/3202185.3202748

[29] Brenda Cantwell Wilson and Sharon Shrock. 2001. Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education* (Charlotte, North Carolina, USA) *(SIGCSE '01)*. Association for Computing Machinery, New York, NY, USA, 184–188. https://doi.org/10.1145/364447.364581

[30] Stephanie Woessner and Niall McNulty. 2021. How the best teachers use bloom's taxonomy in their digital classrooms. https://www.niallmcnulty.com/2017/11/blooms-digital-taxonomy/

[31] Colleen Carraher Wolverton, Brandi N. Guidry Hollier, and Patricia A. Lanier. 2020. The Impact of Computer Self Efficacy on Student Engagement and Group Satisfaction in Online Business Courses. *Electronic Journal of e-Learning* 18, 2 (Feb. 2020), 14. https://doi.org/10.34190/ejel.20.18.2.006

[32] Casey Wong, Paul Denny, Andrew Luxton-Reilly, and Jacqueline Whalley. 2021. The Impact of Multiple Choice Question Design on Predictions of Performance. In *Australasian Computing Education Conference* (Virtual, SA, Australia) *(ACE '21)*. Association for Computing Machinery, New York, NY, USA, 66–72. https://doi.org/10.1145/3441636.3442306