Toward A Framework for Formative Assessment of Conceptual Learning in K-12 Computer Science Classrooms

Shuchi Grover
Looking Glass Ventures/Stanford University
Palo Alto, CA, USA
shuchig@cs.stanford.edu

ABSTRACT

Unlike summative assessment that is aimed at grading students at the end of a unit or academic term, formative assessment is *assessment **for** learning*, aimed at monitoring ongoing student learning to provide feedback to both student and teacher, so that learning gaps can be addressed during the learning process. Education research points to formative assessment as a crucial vehicle for improving student learning. Formative assessment in K-12 CS and programming classrooms remains a crucial unaddressed need. Given that assessment for learning is closely tied to teacher pedagogical content knowledge, formative assessment literacy needs to also be a topic of CS teacher PD. This position paper addresses the broad need to understand formative assessment and build a framework to understand the what, why, and how of formative assessment of introductory programming in K-12 CS. It shares specific programming examples to articulate the cycle of formative assessment, diagnostic evaluation, feedback, and action. The design of formative assessment items is informed by CS research on assessment design, albeit related largely to summative assessment and in CS1 contexts, and learning of programming, especially student misconceptions. It describes what teacher formative assessment literacy PD should entail and how to catalyze assessment-focused collaboration among K-12 CS teachers through assessment platforms and repositories.

CCS CONCEPTS

• **Social and professional topics → Student assessment**; **K-12 education**.

KEYWORDS

formative assessment, computer science education, K-12 computer science education, teacher preparation, introductory programming

ACM Reference Format:
Shuchi Grover. 2021. Toward A Framework for Formative Assessment of Conceptual Learning in K-12 Computer Science Classrooms. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432460

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432460

1 INTRODUCTION

With CS becoming more widespread in school settings, curriculum, tools, and classroom implementation are making steady progress. Teacher preparation has begun to evolve from its initial focus on training new teachers–who often have little to no background in programming or teaching computing–in becoming familiar with the basics of programming, usually in the programming language they were expected to use and the curriculum they are expected to teach. As CS teacher self-efficacy increases, attention is shifting to helping teachers understand pedagogy–the ***how*** of teaching CS. Scaffolding strategies, unplugged activities, creative coding, pair programming are becoming part of teacher professional development (PD). Meanwhile the literature on student difficulties in learning programming, even in easy-to-use block-based environments, continues to grow, as is the large body of literature on addressing novice programmer misconceptions that transcend age, context, and even programming environments. Formative (or classroom) assessment–aimed at *assessment **for** learning*, and often targeting student misconceptions–is a critical omission from K-12 CS education discourse and practice. Studies have identified huge gaps in formative assessment and assessment literacy for K-12 CS teachers [49, 50, 57], even when the evidence suggests that *attention to classroom formative assessment can produce greater gains in student achievement than any other change in what teachers do* [54].

This position paper makes a case for embracing formative assessment in K-12 CS classrooms and 'formative assessment literacy' for CS teachers with the goal of improving teaching and learning of CS, especially introductory programming. It presents a vigorous argument for why this is an urgent and essential need. It also presents dimensions of a framework to help understand the what, why, and how of formative assessment in K-12 CS and guide progress on three key aspects of formative assessment in K-12 CS: formative assessment design, formative assessment literacy in teacher PD, and leveraging community and community-developed resources. CS in K-12 is young, but the time is ripe to give formal formative assessment in K-12 CS much-needed attention.

The following section shares key aspects of formative assessment and why it is the urgent need of the hour for K-12 CS education and CS teacher preparation. Section 2 describes relevant CS education assessment research. Section 3 details dimensions of a framework to guide the integration of formative assessment for better conceptual learning in K-12 CS classrooms including design of assessments, teacher PD, and community collaboration and items repositories.

1.1 The what and why of formative assessment

Put simply, formative assessment is *assessment **for** learning*, or measurement that helps monitor student learning and adjust teaching

accordingly. Seminal papers and groundbreaking research in education around 30 years ago argued for and demonstrated that formative assessment improves student learning [8, 39]. Since then, disciplinary-based education research in all core subjects has paid much attention to understanding and implementing formative assessment in classroom teaching to improve student learning. Formative assessment literacy is closely related to teachers' pedagogical content knowledge (PCK) [37] and should be an imperative for teacher preparation. Google scholar search results on formative assessment in school education run into hundreds of thousands of articles. Only a handful of these are situated in K-12 CS contexts. This section helps build a foundational understanding of formative assessment based on decades of education research.

Formative assessment is ALL about feedback–to the teacher and the learner. Formative assessment is not complete until it has resulted in some follow-on action on the part of the teacher (or teaching agent) as feedback that is internalized by the student. It can be viewed as a linear process, but in reality it is more cyclical—from **monitoring** (*Is learning taking place?*) to **diagnosis** (*What is not being learned?*) to **action** (*What to do about it?*) [9]]. This all-important feedback element of formative assessment is also related to metacognition that helps with learner self-regulation and helping students to "learn how to learn" [29].

Formative assessment is NOT for giving students a grade The word 'test' is an anathema to some teachers [37]. However, the reality of formative assessment could not be farther from norm-referenced and standardized testing that the K-12 CS community largely eschews. The function of the formative assessment process is to supply evidence that will enhance students' learning [53] through feedback and appropriate follow-up instructional moves. It is thus an important piece of the learning puzzle regardless of grade level, programming environment, or the pedagogy used to teach CS. As Bloom said, "we see [much more] effective use of formative evaluation if it is separated from the grading process and used primarily as an aid to teaching (p. 48)"[10].

Formative assessment is used to share learning goals with students A key role of formative assessment is to clarify and share with learners learning intentions and criteria for success, that are originally a teacher's goals. If improvement in learning is to take place, students need to come to hold a concept of quality in line with that held by the teacher, and the community (via standards, for example). This "growing concept of what good work is forms part of the learning itself" [11]. The need for formative assessments is underscored even in classrooms using project-based learning; the best project-based approaches use a combination of ongoing formative assessment and project rubrics that can both communicate high standards and help teachers make judgments about the multiple dimensions of project work [5].

Formative assessments should target and address misconceptions 'Diagnostic questions' (or items) often target student misconceptions. What makes a diagnostic item particularly formative is that an incorrect response not only provides information about gaps in student understanding; it also provides insight into what it is that the student does not understand—in other words, the nature of their misconceptions [13]. The misconceptions literature in introductory programming at all education levels is large and growing. Sorva details several misconceptions in K-12 learners based on research of novice programming since the 1980s [23, 47]. Ideally, formative assessments could and should be targeting these misconceptions through diagnostic items (see section 3.1.1).

Formative Assessment and Teacher PCK Professional learning of formative assessment practices, or formative assessment literacy, also helps build teachers' CS knowledge and PCK; the cultivation of teacher expertise to effectively implement formative assessment is grounded in three core, interrelated domains: formative assessment knowledge and skills, disciplinary knowledge, and habits of practice [27]. Knowledge of misconceptions and how to address them forms the link between PCK and formative assessment. A teacher cannot know whether students harbor naïve or wrong notions of a concept, even after it has been taught, unless a teacher gets feedback through formative assessments.

Formative assessments come in many forms Formative assessment refers to formal and informal moves that teachers employ to make inferences about what their students know and can do during their routine classroom learning. "CS teachers teachers can informally assess students in several ways– a show of hands in response to a question; students' expressions of frustration, disengagement, or joy during a coding task; and informal conversations with students as they code or debug their programs" [26]. CS education research suggests that multiple forms–or "systems"–of formal assessment are also needed to get a holistic view of student learning [21]. These include closed and open-ended programming assignments (that are ideally accompanied by peer and self assessment, student and teacher rubrics, and student reflections), quizzes with multiple choice questions (MCQ), as well as assessments beyond MCQ such as Parson's problems [36], hotspot and point & click items, and unit-test coding assignments that are innovate and also autogradable [26]. These varied forms of formal assessment help address conceptual and affective goals of teaching CS. Given the many known conceptual difficulties of learning programming, a focus on formative assessments that provide speedy and timely feedback on *conceptual learning* is a crucial need. Programming projects are notoriously time-consuming and subjective to score and give feedback on. Even if programs are auto-analyzed - targeted feedback to students is challenging, and the presence of constructs or correct code does not necessarily equal understanding [40].

Speed and timeliness of feedback matters Formative assessment can happen at different time scales. Research shows, however, that teachers' day-to-day classroom practices with an explicit focus on short-cycle assessment are most impactful [53]. When teachers want to quickly survey student thinking, an MCQ has utility in terms of taking little time to ask, collect responses and process them [56]. Thus, MCQ or other interactive, instantly autogradable items designed to be administered as short, quick quizzes embedded in interactive textbooks, or through homework systems or assessment platforms that allow for autograding, are valuable formative tools.

Formative and summative assessment have different designs Summative assessment is assessment **of** learning and happens at the end of a unit, term, or year. Its purposes and goals are fundamentally different from formative assessment, where the goal is to strengthen student understanding **during** learning. Formative assessments need to be diagnostic and focused on misconceptions and what students find historically difficult, or based on granular learning progressions to provide a clear signal of next steps.

2 RELEVANT PRIOR WORK

Assessment has been an active area of research in CS education. However the overwhelming majority of literature is devoted to measuring learning in undergraduate contexts (CS1 classrooms). Research in primary and secondary schools (or K-12) has largely been focused on summative assessment design of assessments for computational thinking (CT) and introductory programming [20, 22, 35, 45, 52], or rubrics for analyzing projects (e.g. [12, 24]). Literature on formative assessment specifically in K-12 CS is limited. Formative assessments are listed as part of "systems of assessments" designed for CS in middle grades [21]. Formative assessment in K-12 have often been the focus of automated tools [6, 34, 51] that are (a) not generalizable as they are restricted to a specific programming environment, (b) provide little guidance on specific areas of difficulty or misconceptions, and (c) may not be completely accurate in truly assessing student understanding [40]. Given the intertwined nature of learning and formative assessment, CS literature on learning progressions [31, 38], models of program comprehension [43], MCQ and other forms of assessments (e.g., [16, 32, 36]) are releveant to formative assessment design, as are assessment taxonomies such as SOLO and Bloom (e.g., [14, 33, 48]) and the large body of research on misconceptions (e.g.,[46, 47]). There is some prior and ongoing work on creating repositories of assessments (mainly MCQ and other autogradable items) [1, 2, 19, 41] that is also relevant. A recent paper focused broadly on assessment for K-12 CS identified need for a taxonomy of assessment in the primary grades, measurements of student progression and growth over time, and creating culturally relevant evaluations and assessment [50].

3 FORMATIVE ASSESSMENT IN K-12 CS

This section outlines key dimensions of a framework that can guide the use of formative assessment in K-12 CS: design of formative assessments, teacher preparation, and community resources and collaboration for formative assessments. These, along with their sub-elements are described in detail along with guidelines drawn from the large body of literature on formative assessment in education research (shared in Section 1.1), as well as analysis of formative assessments in existing (high school) CS curricula in the US, designed CS assessments created by researchers (albeit aimed primarily at summative assessment) [20, 22, 35, 45, 52], and ongoing work in creating corpuses of formative assessments.

3.1 Design of formative assessment items

When designing formative assessment items we need to keep in mind the several goals of formative assessment. For the teacher, the main goals relate to monitoring and diagnosis in order to inform formative action. For the student, formative assessments help:

(1) Establish and maintain an orientation to learning goals (and national or state standards);
(2) Demonstrate to students how to achieve those learning goals;
(3) Highlight key concepts that they might otherwise overlook;
(4) Help to control frustration and confusion (as is often the case among novice programmers [30]); and
(5) Help close the gap between their understanding and the goal.

Formative assessments for K-12 CS classrooms could be formal or informal and target conceptual and affective learning goals. Due to space constraints, and based on the rationale (in Section 1.1) on using autograded assessment types for speedy feedback on conceptual understanding, this paper focuses mainly on formal, designed formative quiz-like check-ins. These are strategic, targeted, autogradable, frequent, low-stakes, and provide quick feedback and explanation. Such items are suitable for probing understanding of key programming concepts (such a sequence, loops, conditionals, functions, expressions, variables and other data structures) and CT practices (such as debugging, problem decomposition, algorithmic thinking, pattern recognition, and abstraction). These need not, however, always involve a code snippet or programming language.

3.1.1 Multiple Choice Question (MCQ) and innovative assessment types. Contrary to what some believe, well-designed MCQ and easily gradable fixed answer types can probe and shine a light on conceptual understanding, and surface student difficulties and gaps in understanding [16, 32]. Past research in CS education has identified various kinds of good question types [43]. These, along with additional ones added by the author, are presented in Table 1.

Question Type	Description/Example
Fixed code	Manually trace through some code and select the correct outcome or result from a set of options
Determine correctness	Given a goal, determine whether a code snippet achieves the goal (requires code tracing)
Compare solutions	Given two or more solutions, pick correct option; or evaluate which is better based on given criteria
Specify variable value	Trace code to determine what the value of variable(s) at a specified point or at the end
Skeleton code	Requires selection of code (from a set of options) that completes the provided "skeleton" code,
Change in logic	Given a code fragment, select from options the code fragment(s) that should give the same result but the logic of the algorithm has been altered (or reversed).
Change in representation	Given an algorithm in pseudo code (or natural language) translate the logic into code in language X (or vice versa).
Code purpose	Given a code segment, explain the purpose of that piece of code in plain English (or select from options)
Code refactoring	Given a code snippet, select options for refactoring or click on code chunks suitable for refactoring.
Parson's problems	Given a goal, rearrange blocks (of code) to achieve the given goal
Debug/Fix Code	Given a goal, identify bug by selecting from options or clicking on blocks or lines of code; or selecting what would fix the code
Code intent	From a test case or series of test cases, determine the intent, the code for which this test specifies the functional intent.

Table 1: Item types for programming (adapted from [43])

Thanks to the affordances of technology, new and more engaging forms of autogradable assessment that provide quick feedback have emerged. New-age assessment platforms provide a rich pallette of new assessment types beyond the traditional multiple choice or multiple answer or fixed response type that make autogradeable assessment more engaging and less taxing cognitively. Parson's

problems [36], created as drag-drop items, have been shown to be a helpful formative learning and assessment activity for novice learners [16, 17]. Additionally, "hotspot" and "point & click" items, are not only well-suited for block-based programming code segments, but they also reduce cognitive load imposed on learners who otherwise might have to keep track of labeled code segments presented as MC options as in Figure 1. Newer browser-based assessment tools also allow for microworlds or in-browser code writing that can be auto-graded based on fixed possible solutions or unit-tests.

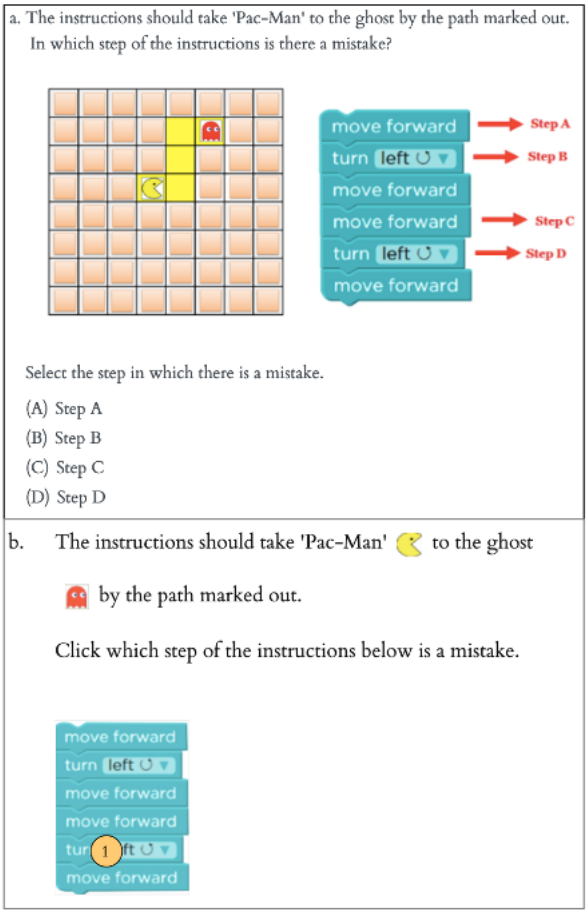


Figure 1: An MCQ item from CTt [20] adapted into a point-and-click item (more intuitive and lower cognitive barriers).

3.1.2 Learning targets. Formative check-ins can target knowledge of the syntax of a programming environment or knowledge of programming vocabulary (e.g. definition of the term "algorithm"). However, much more valuable for teacher feedback and student learning are the ones that target understanding of programming concepts and constructs–how and when to use them, program comprehension–tracing code to figure things out to answer a question, or debugging a code snippet. Formative assessments could also target a known misconception or a key learning goal that is a building block in a learning trajectory of programming.

Targeting misconceptions and known difficulties through diagnostic items. What makes diagnostic items particularly formative is that an incorrect response to a diagnostic item not only provides information that a student does not clearly understand a particular topic; it also provides specific insight into *what it is that the student does not understand*—in other words, the nature of their misconceptions (see Figure 2). There are about 40 to 50 well-known misconceptions in introductory programming relevant to primary and secondary school CS (e.g. [46, 47]), most of which transcend programming environment. Creating a bank of "misconceptions-oriented" items across various grade levels would be a valuable resource for teachers. These items would also serve as crucial training on programming PCK for teachers who are new to teaching CS (see Section 3.2).

Targeting learning progressions, building blocks, and program comprehension. Formative assessment is closely related to learning goals–in the moment and on the day. As such, it is more important that it be informed by what we know about the systematic building of student understanding of programming as articulated in learning trajectories and progressions [3, 31, 38], rather than by standards [4] (national or state-specific) that are oriented more toward summative understanding or learning goals at the end of each grade or grade band. In-the-moment formative assessments must therefore target more granular learning goals of programming that serve as building blocks toward more comprehensive understanding, which is the target of summative assessment. As examples in Figure 2 show, formative assessments can query student understanding of single concepts especially when a concept is first introduced. Schulte's Block Model [42] would suggest that these understandings are at the Atom or Block levels rather than the Relations or Macro-structure levels, although the latter should be part of formative understanding in secondary school. Given that learning of programming is intertwined with program syntax and semantics, it is also important that formative assessments target learning goals that encompass both structure and function as defined in the Block Model (rather than only a learning goals-oriented trajectory such as [38]). Bloom's taxonomy [48] and SOLO taxonomy [7] have been used extensively in tertiary CS education assessment research [14, 33, 44] and could similarly provide guidance on design of formative assessment items target varying levels of program comprehension and CT practices such as debugging, algorithmic thinking, and abstraction.

3.2 Formative assessment literacy for teachers

K-12 CS teachers' lack of confidence or knowledge and skills has impacted the implementation of assessments and the depth of feedback they provide [50]. It is therefore crucial to develop teachers' capacity and influence their habits of practice to make formative assessment integral to their teaching. Teachers need to incorporate the following formative assessment process (adapted from the Michigan Assessment Consortium resources) in their classroom.

(1) Establish clear learning goals and success criteria for lessons, and ensure students understand what these mean and entail;
(2) Plan for and elicit evidence of learning during or in between lessons, and interpret that evidence–as close to the actual

Problems targeting misconceptions	Concepts targeted	What does the student not understand?	What are possible next moves for the teacher?
 What is the value of the variable steps after these two blocks are executed? A. 0 B. 10 C. 20 Many students respond with 20 as the answer	Variable assignment	S does not understand that only set/change blocks will affect the value of a variable	Share examples (a) with variable inspection when variable values change; (b) of how expressions evaluate to a value
Which scripts do exactly the same thing?  (A) A and B (B) B and C (C) A and C (D) None of them do exactly the same thing (E) They all do exactly the same thing	Simple loops (targets "repeating unit" misconception [23])	They do not understand that the commands in a loop repeat as a repeating unit	Examples that trace and "unfurl" a loop Multiple examples with different "repeating units" that help visualize the execution of a group of commands in various ways (sound, print/say, costume chance) VELA "graphical looping" activity

(A)
`number=1`
`print('start')`
`while(number < 10:`
`  number += 4`
`  print(number)`
`print('stop')`
 	
(B)
`number=1`
`print('start')`
`while(number < 10:`
`  print(number)`
`  number += 4`
`print('stop')`
(1) Do A and B print the same values?
(2) What are the numbers printed in each?
(3) What is the value of 'number' after the loop?
	How while loops work; how variables are update; how variable expressions control loops [46]	Some students believe the while loop is continuously checked.	Have students trace the code and write down values for both and compare behaviors.

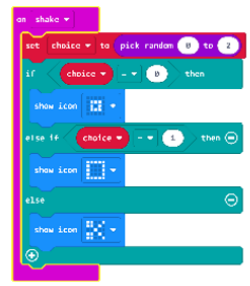
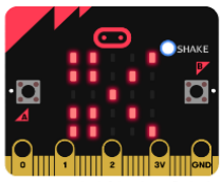
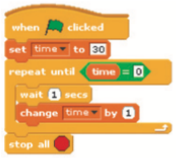
Problems using learning trajectories and using building blocks of comprehension (Block Model)	**Concepts targeted**	**What does the student not understand?**	**What are possible next moves for the teacher?**
  The program on the left has been flashed onto the micro:bit. When the micro:bit is shaken, it displays the icon shown above. **What value is stored in the choice variable?**	Nested If-Else statements	How control flow works in code with nested IF-Then-Else statements	Break it down into a simple IF-Else conditional first and demonstrate control flow. Then add the nested IF-THEN and step-by-step help trace the code to see what the 'K' suggests about the value in the 'choice' variable
Raul wants to make a timer that will count down from 30 to 0. Raul has written the following code using a time variable:  (1) Will Raul's code work as desired? Yes / No (2) In Raul's code, will the Repeat Until loop eve stop (i.e., will the "time=0" condition ever b satisfied?)? Yes / No (3) If you had to change *just one thing* to fix the bug, what would you change? ❑ The `Set` time block ❑ The `Repeat Until` "time=0" condition ❑ The `wait` block ❑ The `change` time `by` block ❑ The `stop all` block	- Controlling a loop with a variable [38] - Configuring a condition to stop the loop - Variable initialization and updating	Break down the concepts to isolate problem from among the possible ones.	This question should not be given as a formative assessment in primary or middle grades as it addresses the relational level of the Block Model.

Figure 2: Formative assessment design. Examples based on research on misconceptions, learning trajectories, and levels of program comprehension. Process of assessment, diagnosis and formative action described.

time of the lesson as possible–to judge where students are in relation to learning goals and success criteria;
(3) Take pedagogical action based on evidence of learning and provide students feedback linked to learning goals and success criteria. Feedback during lessons helps to scaffold students' learning by helping them to answer: *Where am I going? Where am I now? What are my next steps?*
(4) Support students to engage in peer- and self-assessment and self-reflection in addition to other quick means of feedback in order to strengthen their awareness, collaboration, confidence, efficacy, and autonomy as learners; and
(5) Foster a collaborative classroom culture where students and teachers are partners in learning.

In order to make the formative assessment integral and meaningful to the CS teaching and learning process, CS teachers also need to know and understand (a) what formative assessment looks like in an introductory CS classroom– what are informal and formal forms of monitoring student progress? (b) how assessments should be incorporated in CS classrooms– when should a quiz be administered and how? A single diagnostic item given to the entire class in the middle of a whole class discussion? or an exit ticket at the end of a period? or a review at the beginning of the next period? (c) the assessment item itself– what is the item targeting? what does a wrong answer suggest? what follow-up action should be taken? (Fig. 2). In addition, they also need to know strategies of formative action in various situations– a single student or a few students providing an incorrect response will require different action than a when sizeable proportion of the class responds incorrectly to a certain question (e.g. [25]). (d) how to design their own assessments. In addition, there is a need for community collaboration and support that works to provide teachers access to shared banks of exemplary assessments for the target topics/concepts or learning goals at the grade(s) they are teaching, in the desired programming language, and at varied difficulty level(s) for their students (Section 3.3).

To be successful, CS teacher assessment literacy must concentrate on both content and process; but we must first focus on **what** we want teachers to change about what they do, and ***then*** work on **how** to support teachers in making those changes. This is key, because students benefit *only when teachers change what they do in classrooms* (and not based on what teachers think) [54]. Additionally, we need to build assessment measures of teacher assessment literacy that take into account the introductory CS context and include factors shown to influence student assessment [15].

3.3 Assessment repositories, feature-rich platforms, and community collaboration

Teacher learning communities are a powerful mechanism to improve teachers' capabilities in using assessment in the service of learning [55]. Additionally, a teacher community of practice often sustains itself around a shared need and the give and take of shared resources for all to benefit [28]. Anecdotally and empirically [49], it is evident that teachers are in need of assessments and especially, formative assessment-oriented items designed with care and purpose (based on the guidelines in Section 3.1) to aid their teaching classroom assessment efforts. Fincher [18] outlined six criteria for successful repositories in CS education: **Control** (who can contribute, who can use it, and how is this enforced?); **Contributors** (how are people motivated to contribute? how are they rewarded?); **Catalogue** (how will users find what they are looking for in the repository?); **Community** (is there a community of contributors? of users? how are these communities built and sustained, and what roles do they play?); **Curation** (how is the data maintained?) and **Content** (are the items in the collection focused on a specific topic or more general purpose? is the content based on a pre-existing collection or created specifically for the repository (or both)?) Extant and ongoing efforts for CS assessment item banks and repositories include Edfinity, Project Quantum, Viva, and the Canterbury Question Bank (focused on CS1) [1, 2, 19, 41].

Assessment platforms and homework systems can serve as item repositories for aggregation, creation, curation, and cataloging or taxonomizing of assessments based on multiple and multi-level taxonomies including, among others, CS/CT topics, learning standards, grade, difficulty level, programming language, and ad-hoc metadata to support easy search and discovery. These technology platforms can also aid with assessment delivery, administration, auto-grading, and teacher dashboards. Back-end data and analytics on student performance can provide teachers crucial insights into students' learning and understanding at individual and aggregate levels. Such technology platforms should be affordable and afford features important for formative assessment such as provision for multiple attempts of a question by students, hints, and feedback (or explanation) for the correct and incorrect options. Solution explanation must be provided as feedback. These explanations, as also the question stem should support rich text, graphics and video for better learner engagement and multiple modes (and languages) of presentation to equitably support diverse learners. Item banks must include technology-enhanced assessments that push the boundaries to include interactivity, drag-drop (for items types such as Parson's problems), microworlds, and in-browser code entry and testing. Furthermore, technology platforms could innovate with randomized variants of items, solution validation, and customized feedback. It is imperative that assessment aggregation also support features for teacher collaboration, contribution, attribution, and sharing, as well as interfaces for creation of both simple and technology-enhanced items. Paper formative assessments cannot be autograded or leverage aforementioned affordances of technology. Tools such as Google Forms, while popular for formative assessment, do not auto-grade or afford many of the features described above.

4 CONCLUSION

This position paper makes a persuasive argument for formative assessment and teacher formative assessment literacy in K-12 CS, keeping the goal of robust student learning in mind. CS is more than programming and conceptual learning. However, through focusing the framework and its dimensions on conceptual learning and examples of formative assessment forms, along with designs, tools, and guidance for providing convenient and powerful formative feedback, this paper makes a start in addressing a crucial lacuna.

5 ACKNOWLEDGEMENTS

This material is based upon work supported by NSF #1943530. Thanks also to the CS teachers partnering on this work.

REFERENCES

[1] Edfinity. https://edfinity.com. Accessed: 2020-08-25.

[2] Project quantum. https://diagnosticquestions.com/Quantum.

[3] M. Armoni. Spiral thinking: K–12 computer science education as part of holistic computing education. *ACM Inroads*, 5(2):31–33, 2014.

[4] C. S. T. Association et al. Csta k-12 computer science standards, revised 2017. *Computer Science Teachers Association, USA*, 2017.

[5] B. Barron and L. Darling-Hammond. How can we teach for meaningful learning? In L. Darling-Hammond et al., editors, *Powerful learning: What we know about teaching for understanding*, pages 11–16. Jossey-Bass, San Francisco, 2008.

[6] A. Basawapatna, A. Repenning, and K. H. Koh. Closing the cyberlearning loop: Enabling teachers to formatively assess student programming projects. In *Proceedings of the 46th SIGCSE*, pages 12–17. ACM, 2015.

[7] J. B. Biggs and K. F. Collis. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 2014.

[8] P. Black and D. Wiliam. Assessment and classroom learning. *Assessment in Education: principles, policy & practice*, 5(1):7–74, 1998.

[9] P. Black and D. Wiliam. Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education)*, 21(1):5, 2009.

[10] B. S. Bloom. Some theoretical issues relating to educational evaluation. *Educational evaluation: new roles, new means: the 63rd yearbook of the National Society for the Study of Education*, 69:26–50, 1969.

[11] S. M. Brookhart. Developing measurement theory for classroom assessment purposes and uses. *Educational measurement: Issues and practice*, 22(4):5–12, 2003.

[12] V. Cateté, N. Lytle, and T. Barnes. Creation and validation of low-stakes rubrics for k-12 computer science. New York, NY, USA, 2018. ACM.

[13] J. Ciofalo and C. E. Wylie. Using diagnostic classroom assessment: one question at a time. *Teachers College Record*, 108(1), 2006.

[14] T. Clear, J. Whalley, R. Lister, A. Carbone, M. Hu, J. Sheard, B. Simon, and E. Thompson. Reliably classifying novice programmer exam responses using the solo taxonomy. *National Advisory Committee on Computing Qualifications*, 2008.

[15] C. DeLuca, A. Valiquette, A. Coombs, D. LaPointe-McEwan, and U. Luhanga. Teachers' approaches to classroom assessment: A large-scale survey. *Assessment in Education: Principles, Policy & Practice*, 25(4):355–375, 2018.

[16] P. Denny, A. Luxton-Reilly, and B. Simon. Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research*, pages 113–124, 2008.

[17] B. J. Ericson, L. E. Margulieux, and J. Rick. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, pages 20–29, 2017.

[18] S. Fincher, M. Kölling, I. Utting, N. Brown, and P. Stevens. Repositories of teaching material and communities of use: nifty assignments and the greenroom. In *Proceedings of the Sixth ICER*, pages 107–114, 2010.

[19] D. Giordano, F. Maiorana, A. P. Csizmadia, S. Marsden, C. Riedesel, S. Mishra, and L. Vinikienė. New horizons in the assessment of computer science at school and beyond: Leveraging on the viva platform. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 117–147. 2015.

[20] M. R. González. Computational thinking test: Design guidelines and content validation. In *Proceedings of EDULEARN15 conference*, pages 2436–2444, 2015.

[21] S. Grover. Assessing algorithmic and computational thinking in k-12: Lessons from a middle school classroom. In *Emerging research, practice, and policy on computational thinking*, pages 269–288. Springer, 2017.

[22] S. Grover. Designing an assessment for introductory programming concepts in middle school computer science. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 678–684, 2020.

[23] S. Grover and S. Basu. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 267–272, 2017.

[24] S. Grover, S. Basu, and P. Schank. What we can learn about student learning from open-ended programming projects in middle school computer science. In *Proceedings of the 49th SIGCSE*, pages 999–1004, 2018.

[25] S. Grover, R. Pea, and S. Cooper. Promoting active learning & leveraging dashboards for curriculum assessment in an openedx introductory cs course for middle school. In *Proceedings of the 1st ACM Learning@scale*, 2014.

[26] S. Grover, V. Sedgwick, and K. Powers. Feedback through formative check-ins. In S. Grover, editor, *Computer Science in K-12: An A to Z Handbook on Teaching Programming*. Edfinity, 2020.

[27] M. Heritage and C. Wylie. Reaping the benefits of assessment for learning: Achievement, identity, and equity. *ZDM*, 50(4):729–741, 2018.

[28] C. Hoadley. What is a community of practice and how can we support it? *Theoretical foundations of learning environments*, 286, 2012.

[29] J. Hudesman, S. Crosby, B. Flugman, S. Issac, H. Everson, and D. B. Clay. Using formative assessment and metacognition to improve student achievement. *Journal of Developmental Education*, 37(1):2, 2013.

[30] M. Israel, Q. M. Wherfel, J. Pearson, S. Shehab, and T. Tapia. Empowering k–12 students with disabilities to learn computational thinking and computer programming. *TEACHING Exceptional Children*, 48(1):45–53, 2015.

[31] C. Izu, C. Schulte, A. Aggarwal, Q. Cutts, R. Duran, M. Gutica, et al. Fostering program comprehension in novice programmers - learning activities and learning trajectories. New York, NY, USA, 2019. ACM.

[32] R. Lister. One small step toward a culture of peer review and multi-institutional sharing of educational resources: a multiple choice exam for first semester programming students. In *Conferences in Research and Practice in Information Technology Series*, 2005.

[33] R. Lister, B. Simon, E. Thompson, J. L. Whalley, and C. Prasad. Not seeing the forest for the trees: novice programmers and the solo taxonomy. *ACM SIGCSE Bulletin*, 38(3):118–122, 2006.

[34] J. Moreno-León, G. Robles, and M. Román-González. Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46):1–23, 2015.

[35] A. Mühling, A. Ruf, and P. Hubwieser. Design and first results of a psychometric test for measuring basic programming abilities. In *Proceedings of the workshop in primary and secondary computing education*, pages 2–10, 2015.

[36] D. Parsons and P. Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 157–163, 2006.

[37] W. J. Popham. Assessment literacy for teachers: Faddish or fundamental? *Theory into practice*, 48(1):4–11, 2009.

[38] K. M. Rich, C. Strickland, T. A. Binkowski, C. Moran, and D. Franklin. K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM ICER*, pages 182–190, 2017.

[39] D. R. Sadler. Formative assessment and the design of instructional systems. *Instructional science*, 18(2):119–144, 1989.

[40] J. Salac and D. Franklin. If they build it, will they understand it? exploring the relationship between student code and performance. In *Proceedings of the 2020 ACM ITiCSE*, pages 473–479, 2020.

[41] K. Sanders, M. Ahmadzadeh, T. Clear, S. H. Edwards, M. Goldweber, C. Johnson, R. Lister, R. McCartney, E. Patitsas, and J. Spacco. The canterbury questionbank: building a repository of multiple-choice cs1 and cs2 questions. In *Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports*, pages 33–52, 2013.

[42] C. Schulte. Block model: An educational model of program comprehension as a tool for a scholarly approach to teaching. ACM, 2008.

[43] C. Schulte, T. Clear, A. Taherkhani, T. Busjahn, and J. H. Paterson. An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports*, pages 65–86. 2010.

[44] C. C. Selby. Relationships: computational thinking, pedagogy of programming, and bloom's taxonomy. In *Proceedings of the workshop in primary and secondary computing education*, pages 80–87, 2015.

[45] E. Snow, D. Rutstein, M. Bienkowski, and Y. Xu. Principled assessment of student learning in high school computer science. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 209–216, 2017.

[46] E. Soloway and J. C. Spohrer. *Studying the novice programmer*. Psychology Press, 2013.

[47] J. Sorva. Naive conceptions of novice programmers. In S. Grover, editor, *Computer Science in K-12: An A to Z Handbook on Teaching Programming*. Edfinity, 2020.

[48] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 155–161, 2008.

[49] R. Vivian and K. Falkner. A survey of australian teachers' self-efficacy and assessment approaches for the k-12 digital technologies curriculum. In *Proceedings of the 13th WiPCE*, pages 1–10, 2018.

[50] R. Vivian, D. Franklin, D. Frye, A. Peterfreund, J. Ravitz, F. Sullivan, M. Zeitz, and M. M. McGill. Evaluation and assessment needs of computing education in primary grades. In *Proceedings of the 2020 ITiCSE*, pages 124–130, 2020.

[51] C. G. Von Wangenheim, J. C. Hauck, M. F. Demetrio, R. Pelle, N. da Cruz Alves, H. Barbosa, and L. F. Azevedo. Codemaster–automatic assessment and grading of app inventor and snap! programs. *Informatics in Education*, 17(1):117–150, 2018.

[52] E. Wiebe, J. London, O. Aksit, B. W. Mott, K. E. Boyer, and J. C. Lester. Development of a lean computational thinking abilities assessment for middle grades students. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 456–461, 2019.

[53] D. Wiliam. Formative assessment: Getting the focus right. *Educational assessment*, 11(3-4):283–289, 2006.

[54] D. Wiliam and S. Leahy. Sustaining formative assessment with teacher learning communities. In *PERIHA Professional Learning Series Workshop*, 2012.

[55] D. Wiliam and M. Thompson. Integrating assessment with learning: What will it take to make it work? Routledge, 2008.

[56] E. Wylie and J. Ciofalo. Supporting teachers' use of individual diagnostic items. *Teachers College Record*, 2008.

[57] A. Yadav, D. Burkhart, D. Moix, E. Snow, P. Bandaru, and L. Clayborn. Sowing the seeds: A landscape study on assessment in secondary computer science education. *Comp. Sci. Teachers Assn., NY, NY*, 2015.