Learning Debt in Computing Education

Mr. Nathan H Bean, Kansas State University

Dr. Nathan Bean is a Teaching Associate Professor at Kansas State University Department of Computer Science and Co-Director of the Advancing Learning and Teaching in Computer Science (ALT+CS) Lab. His research is focused on the need to grow the body of students skilled in computing – both within the field of Computer Science, and within other disciplines that increasingly rely on the tools computer science makes available to advance their own work. Thus, his research involves investigations into how to effectively reach a broader and more diverse audience of students, and developing pedagogical techniques and technologies that allow it to be done at scale.

Russell Feldhausen, Kansas State University

Russell Feldhausen received a bachelor's degree in computer science in 2008, and a master's degree in computer science in 2018, both from Kansas State University. He is currently pursuing a doctorate in computer science with a focus on computer science education, also at K-State. Feldhausen's research interest is computer science education, targeting rural populations and exploring ways to integrate mastery learning into CS curricula. He is also actively involved in many K-12 outreach programs providing curricula and teacher training throughout Kansas.

Joshua Levi Weese, Kansas State University

Dr. Josh Weese is a Teaching Associate Professor at Kansas State University in the department of Computer Science. Dr. Weese joined K-State as faculty in the Fall of 2017. He has expertise in data science, software engineering, web technologies, computer science education research, and primary and secondary outreach programs. Dr. Weese has been a highly active member in advocating for computer science education in Kansas including PK-12 model standards in 2019 with an implementation guide the following year. Work on CS teacher endorsement standards are also being developed. Dr. Weese has developed, organized and led activities for several outreach programs for K-12 impacting well more than 4,000 students.

Learning Debt in Computing Education

Abstract

Computer Science is a domain characterized by cumulative learning, where subjects and skills build upon foundational, earlier-learned skills, much like mathematics or human languages. This is especially true for programming, which is often used as a tool to support hands-on learning of more advanced computer science topics. But what happens when there are gaps in a student's foundational learning? This paper explores that question, examining the literature in educational philosophy, psychology, and cognitive science, and offers up a new theoretical concept – learning debt – to help capture the cost and effects of that condition and frame it within the growing emphasis on competency models for designing and evaluating curriculum. Finally, drawing from this understanding, we offer curated suggestions for improving student outcomes emerging from the literature.

1 Introduction

Competency models have been gaining traction in education circles. Representing the interaction of disciplinary knowledge, skills, and practices, *competency* equates to the ability to perform common tasks within a discipline [1]. Educational standards are increasingly shifting to focus on competencies instead of learning outcomes. This is also true for computing disciplines, with the ACM embracing a competency model in its 2020 Computing Curricula recommendations with specifications consisting of knowledge, skills, and dispositions observable in the completion of a task [2]:

$$Competency = [Knowledge + Skills + Dispositions]\ in\ Task$$

The adoption of this model for the ACM/IEEE Computer Science Curriculum Recommendations began in the 2023 cycle with sample competency specifications [3]. This shift in perspective reflects a more holistic view of student learning.

For a discipline in which cumulative learning dominates (like mathematics, language, and computer science) competencies are critical, as within these disciplines new learning builds on prior. For example, developing algebraic reasoning builds upon knowledge and skill with operations such as addition, subtraction, multiplication, and division. In turn, these build upon understanding and ability to use numeric systems, i.e. representing quantities with symbols. Likewise, the general pattern of learning written and oral language generally proceeds from relatively simple to complex. While humans are immersed in complex demonstrations of oral language from birth, in typically developing children, approximations progress in developmentally predictable ways [4, 5]. Although the human drive to make sense, whether of code or print, is innate, reading is not. Gaps in a child's ability to comprehend print can

be theorized socio-culturally as gaps in opportunity [6] and opportunity to learn where input becomes intake [7].

However, language plays an even more vital role in learning, as new knowledge in all fields has been created and disseminated through language. The development of the printing press accelerated human progress, largely by making it possible to capture current knowledge in written form and disseminate it widely. The modern educational system is largely founded on this simple dissemination model, otherwise known as the banking model of education [8]. Knowledge is deposited (or disseminated) into the heads of learners via rote learning, then regurgitated as evidence of mastery. Early grades focus heavily on developing foundational literacy to support later learning efforts in other disciplines. The fourth-grade slump exposes the resultant learning gaps when specialized disciplinary language and more abstract learning replaces the everyday language more common in the primary grades [9]. Latent gaps manifest when new language complexities outrun students' previous learning. Because learners lacked the opportunity to deeply understand language, they often are soon "swamped"[10] and will face significant disadvantages with more complex demands.

Computer science faces a similar challenge. Higher-level learning in computer science often relies on students developing conceptual knowledge by writing programs utilizing the concepts they are learning. In the same sense as the K-12 educational model is built on learning language, then using language to learn, students in an introductory programming sequence (often using the designations CS0, CS1, CS2) are learning fundamentals of programming, and will later apply that fundamental programming knowledge in learning other CS topics. The learning science behind the impact on learning for students struggling to read is well-developed and very clear. However, the computer science education literature has not explored this topic as deeply. In this paper, we seek to lay the foundation for such an exploration, drawing upon contributions from educational philosophy, psychology, and cognitive science, and offer up our own theoretical construct – learning debt – as a useful framing mechanism for considering this impact.

2 Theoretical Foundations For Learning

The concept of cumulative learning arises from Gagné [11] and was expanded upon by Bloom [12, 13]. Perhaps Bloom's best recognized contribution to the field of learning is his Taxonomy, for which he adopted a pyramid shape to emphasize that higher-order thinking builds upon (and requires) foundational knowledge [12]. This taxonomy has been revised several times, and the ACM CCECC developed verb mappings for the Bloom levels specific to computer science [14]. This work was the basis of the CS2023 skill levels as shown in Figure 1 [3].

An explanation for the mechanism underpinning cumulative knowledge arises from the work of Jean Piaget, specifically his theories of genetic epistemology [15]. As a biologist, Piaget observed that a freshwater snail would develop a different phenotype depending on its environment. Move a snail from one lake to another, and it would adapt, becoming more like the snails found there. He theorized that human learning was similar; a process of the brain adapting to its environment – the experiences and knowledge it encountered [16]. He

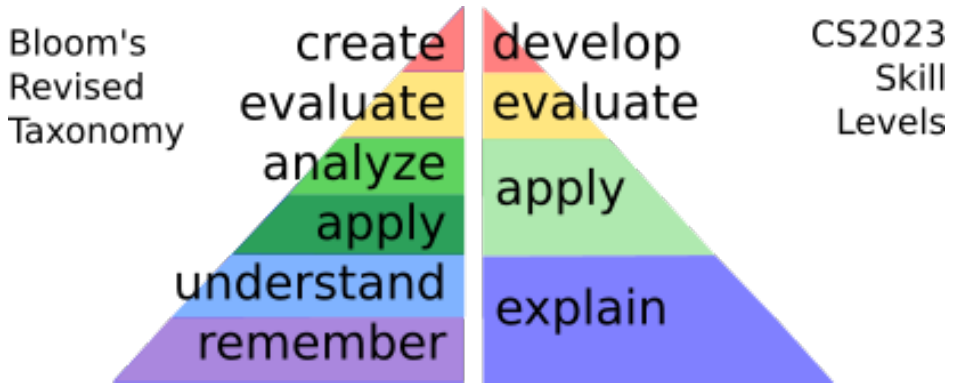

Figure 1: Bloom's Revised Taxonomy and the corresponding CS2023 Skill Levels

suggested two learning mechanisms – assimilation and accommodation. Assimilation referred to knowledge that easily fit into the learners' current understanding of the world, i.e. learning about a new insect when one already had a firm grasp of what an insect is, or adapting to using an aluminum baseball bat when one had learned with wooden ones. In contrast, accommodation required the mind to adapt new structures for reasoning, i.e. learning to tie shoes when one has not learned knots or grasping calculus when one only has experienced algebra. Once accommodation had created these new structures, the topic at hand is suddenly grasped, and problems that previously were a great struggle are now approachable – in effect, successful accommodation is the "eureka" moment.

Piaget theorized this accommodation process needed significant stimulus – which he called mental disequilibrium – to start and maintain building the necessary mental structures [15]. Too little disequilibrium, and the change would not trigger; too much, and the mind would simply reject the new learning. There is a clear parallel here with Vigotsky's zone of proximal development, which is often visualized as a concentric Venn diagram with what a student knows, what a student can learn building from what they already know (the proximal zone), and an area outside that zone that is simply unapproachable to the student at that point [17] (see Figure 2). Good instruction seeks to stay in that zone, not so easy as to be boring, but not so challenging as to be overwhelming, fostering what is often called a productive struggle in teaching practice.

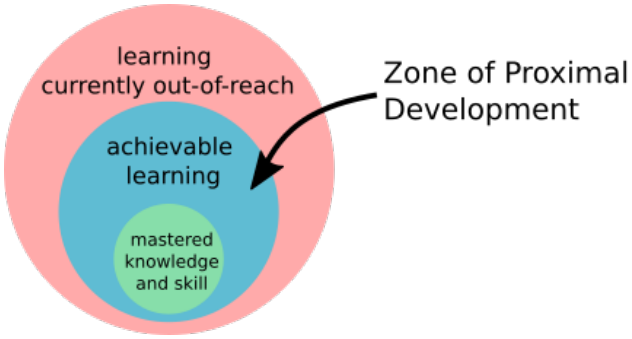

Figure 2: Vygotsky's Zone of Proximal Development

Psychology provides another lens to view this process through cognitive load theory. Its central premise is that the brain has a limited amount of "working memory" that can hold unique items; exceeding this capacity means that something must be discarded for a new item to be considered [18]. A common experience illustrating this phenomenon is when you forget what you came into a room to retrieve because something distracted you on your way. Research has suggested that the average person's working memory can hold only 5-7 things... which begs the question of how we can solve complex problems or carry out complex actions. Cognitive load theory suggests the answer lies in schema – mental structures that allow items to be grouped together and manipulated as a single item in working memory [19]. Hence, developing a schema for addition, subtraction, multiplication, and division becomes necessary for effectively performing algebra. Development of these schema corresponds to Piaget's accommodation process and the expansion of the inner and outer bounds of the zone of proximal development.

However, Piaget is perhaps best known for his learning stages model, focused on how reasoning becomes increasingly abstract as infants learn, moving through four stages: sensorimotor, preoperational, concrete operational, and formal operational [20]. Continuing his work, neo-Piagetians argue that these stages occur with all learning and at all ages, and have espoused an "overlapping waves model" to capture the idea that a learner may be at different stages with some aspects of a particular domain at the same time [21]. For example, a student may be at formal operational with algebraic operations, but concrete operational with exponents and preoperational with factoring algebraic expressions as they first encounter the subject in an algebra class. This connects back to both Piaget's genetic epistemology and cognitive load theory; a learner achieving a new stage corresponds to the development of schema for the subject's concepts and operations.

In their work, neo-piagetians Donna Teague and Raymond Lister have mapped the development of computer programming facility to Piaget's stages, suggesting that the development of the ability to trace code (explain line-by-line) corresponds to the preoperational stage, the ability to write small programs from well-defined specifications to the concrete operational stage, and ability to write a program to solve more open-ended problems corresponding to the formal operational stage [22, 21].

Perhaps what is most important to grasp here is that with each new topic introduced, a student begins in the sensorimotor stage, and must grapple with the topic and develop and incorporate new schema, reflecting a greater understanding and ultimately the achievement of competency.

3 Learning Debt

The recognition that students may move forward in a curriculum without developing mastery (competency) with the subject of a course has significant impacts when viewed through the lens of learning theories we have just explored. In a cumulative knowledge domain, a student without adequate foundational development will find later tasks impossible. Consider expecting a student to read a textbook chapter and write a short essay on what they learned when they have yet to achieve proficiency in reading and writing – a common reality in our

educational system, as 1 in 5 American adults struggle with low literacy skills [23]. But the issue can be more subtle than that. Cognitive load theory has long recognized that the act of learning requires a portion of the available cognitive load of a student (known as the germane load) [18]. If working memory to handle the germane load is not available, then no learning occurs – even if the student accomplishes the task! Thus, it is possible for a student to complete all the assignments in a course and yet fail to expand their ability and comprehension with the subject at hand.

Moreover, this effect snowballs. Consider that a student may be able to pass several courses successfully without gaining the necessary competency. Viewed from the perspective of the overlapping waves model, with each subsequent topic the student has more and more troughs (the low points of the wave) lining up, signifying a greater degree of cognitive load to tackle, and are swiftly moving out of the zone of proximal development into the area of "what a student **cannot** yet do." The increasing pressure to perform, coupled with the difficulty of doing so, leads to one of many undesirable outcomes such as over-reliance on help, resorting to cheating, dropping out, depression, and so on.

We would like to propose the term "learning debt" to encapsulate this idea of missed learning, paralleling the concept of "technical debt" in software development. Technical debt refers to the future costs involved in maintaining code based on decisions made during development that prioritized speed of development and immediate profits over quality and robustness. Likewise, students espousing a "C's earn degrees" mentality, over-reliance on support from peers or generative AI, or those whose circumstances otherwise make it difficult to obtain mastery in a course, build up learning debt by focusing on short-term goals over long-term learning outcomes.

As technical debt builds, it becomes increasingly challenging to modify a codebase, requiring more time (and associated expense) to avoid inadvertently introducing software bugs and security vulnerabilities. Likewise, new functionality built on top of poor design increases the technical debt, and if left unchecked, eventually a codebase may become unmaintainable.

Adopting good development practices can help reduce the amount of technical debt that is created with a new project, though eliminating it entirely is likely impossible. Likewise with student learning – when students adopt good learning practices (ideally encouraged through good pedagogical practices and instructional design), learning debt can be minimized, but some will likely still occur (e.g. a student missing a few lessons because of an illness, failing to internalize some learning due to outside stressors like family issues, an argument with a romantic partner, or other common causes). For software development, the practice of refactoring the code (rewriting portions to improve the design and function of existing features) is commonly employed to reduce technical debt. For a student's learning, the corollary would be targeted and sustained practice with earlier, less-mastered concepts to enhance understanding and develop mastery.

It is also important to note that there are many sources of learning debt beyond the individual student. When a student transfers between programs, such as transferring from a two-year to four-year college, a mismatch between topics covered in the two institutions effectively introduces learning debt with every transferred course, as there will invariably be differences in

coverage of topics between the two institutions. This experience could be equated to porting an existing software project to a new language or framework, which invariably introduces technical debt as the new context will have different preferred implementations.

Likewise, program-wide changes within a single institution can introduce learning debt for the cohort caught in the changeover. And perhaps even more impactful can be missed topical coverage in a properly sequenced course, whether due to outside events (e.g. pandemics), or internal programmatic ones (e.g. a new instructor teaching the course for the first time, an overworked and unprepared researcher asked to cover a core undergraduate course, and so on).

Ultimately, technical debt cannot be avoided entirely – it can only be acknowledged, managed, and mitigated. Likewise for learning debt. By acknowledging its existence, giving careful consideration in how to structure our instruction to minimize its development, and adopting mitigation strategies to assist our students in actively reducing it, we can help improve our outcomes considerably, much as the software industry has learned to handle technical debt. Later sections of this paper will focus on suggestions of how to do so, drawn from existing, research-backed pedagogical and instructional design strategies. But first, it behooves us to briefly explore learning debt's counterpart, competency.

4 Competency Models

Competency models seek to shift from describing curriculum largely in terms of a body of knowledge to be covered to what a successful student should be able to accomplish if they are deemed "competent"; it includes not just knowledge but the effective application of knowledge and skill to carry out tasks, as well as the attitudes and mindset supporting the effort [1]. In this sense they represent a more wholistic view of learning centered around the science of learning previously discussed in section 2.

The competency model developed for the Computing Curricula (CC2020) project and adopted by the ACM Curricula Recommendations (CS2023) incorporates three dimensions, *knowledge* (know-what), *skills* (know-how), and *dispositions* (know-why), situated within a specific *context*. A detailed discussion of this competency model and its development can be found in Frezza et al. [1], but briefly:

- **Knowledge** embodies what students need to know to be competent; it maps directly to the body of knowledge identified in the ACM curriculum recommendations. For programming, knowledge includes syntax and grammar of a language, design patterns, program execution models, etc.
- **Skills** are the ability to utilize that knowledge in context, i.e. to explain a concept, use a control-flow structure effectively, evaluate potential approaches, adapt existing code, and develop original source code.
- **Dispositions** are broader behaviors and attitudes that help a student accomplish a task and learn from it, such as incremental development, investigation, planning, testing, collaboration, reflection, meta-cognition, and perseverance.

While knowledge, skills, and dispositions are often subject to our focus, there are important considerations for the context in which competencies manifest. A context that is perceived as relevant and authentic (i.e. represents the kind of work a student expects to do professionally) can encourage the development and engagement of desirable dispositions. Likewise, a change to the context will affect competency; for example, asking a student to write a program in a different IDE than the one they have always used previously will likely result in a lower competency. Similarly, when the context students are using for homework differs from exams (i.e. collaboration and resources normally available for homework are unavailable for an exam), the competency measured by the exam is likely to be different than that of the students' homework efforts.

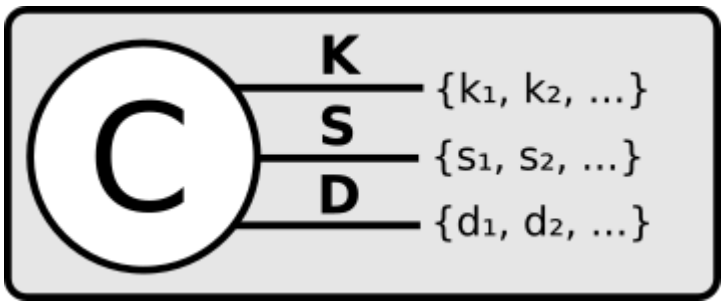

Figure 3: Knowledge, skill, and disposition components of a competency, adapted from [1]

In addition to the competency model, the CC2020 team developed a *competency-based framework for learning* (CoLeaF) which considers each competency as "an integrative function consisting of a set of knowledge elements, a set of skill elements, and a set of disposition elements [1, p.157]," as depicted in Figure 3. These competencies are arranged in a directed acyclic graph representing the dependency relationships between competencies, building from more concrete to abstract understanding, and with the recognition that students may progress through the learning in different orders as illustrated in Figure 4.

Within this framework, learning debt can be understood as unmet prerequisite competencies. For example, if the student has developed competence with C_0 but not C_1, and begins working on a lesson for developing C_3, the learning they are attempting will encompass

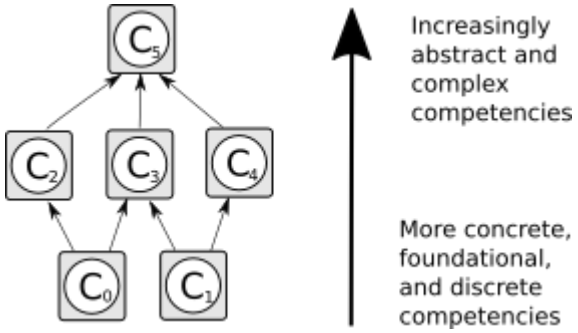

Figure 4: Instance of CoLeaF showing dependencies between competencies from concrete to abstract adapted from [1]

both C_1 *and* C_3. Likewise, if they have only mastered the set of competencies $\{C_0, C_2\}$ and start a homework that targets C_5, they will simultaneously be working on learning the set of $\{C_1, C_3, C_4, C_5\}$. A key strategy to avoid this untenable situation is ongoing formative assessment of each students' competencies.

5 Measuring Competence And Learning Debt

To effectively gauge learning debt (as well as support learning mastery approaches), we need assessments that can provide good estimates of student competence with individual skills and concepts – what testing parlance calls "traits." Statistical approaches like that adopted by *Item Response Theory* (IRT) seek to measure this trait while accounting for item difficulty, likelihood of guessing correctly, and the item's ability to discriminate between students possessing differing amounts of the trait. IRT is also commonly used to validate assessments [24] and to evaluate them for potential bias [25].

IRT has predominantly been used for multiple-choice questions, which have the benefit of being easy to write and score at scale, but also the drawback of predominantly focusing on assessing acquired knowledge. In CS education, the most common form of multiple-choice, knowledge-centric IRT validated assessment are probably *concept inventories*, which use a set of standard questions used to measure student understanding of specific concepts in a discipline [26]. A good example of this type of instrument is the Second CS1 Concept Inventory, which seeks to assess student understanding of foundational programming concepts in a language-independent way [27, 24].

However, IRT-based approaches have also been explored for assessing computer science skills: Muhling et al. used multiple-choice answers with code snippets to investigate students' ability to trace code [28]; Wu et al. used IRT to validate micro Parson's problems [29]; Lai used visualizations and Parson's problems to assess computational thinking [30]; and Berges et al. even developed an approach for using IRT with open-ended code writing problems [31].

Dispositions include well-identified psychometric constructs in the literature with validated assessment instruments including mindsets [32, 33, 34] and professional identity and belonging [35]. Writing "clean" code of high quality (evaluated in terms of readability, understandability, and adherence to norms) is another important disposition; the assessment of which can be done through linters and static code analysis [36, 37, 38]. Instrumenting student code development environments for keystroke-level data collection can likewise provide engagement and editing metrics, which could be analyzed for evidence of iterative program refinement [39, 40, 41].

Bringing these ideas together, Gao et al. present a strategy for assessing student competence through combining multiple assessment strategies into five channels representing measurements of students' *score*, *engagement*, *code metrics*, *programming skills*, and *coding style* carried out sequentially over the duration of a CS1 course [42]. Such a strategy could be expanded to cover a full programming course sequence, and the resulting measures could potentially be shared with students to help drive meta-learning strategies. Such approaches could also be combined with mastery learning (discussed in section 6.2) to inform supplemental instruction.

Finally, IRT can be utilized with *Computer Adaptive Test* (CAT) strategies. In this approach, the difficulty of questions is chosen based on previous answers, allowing the exam to more quickly and precisely measure a student's knowledge, skill, or competence [43]. CAT exams take less time and provide more precise measurements of a student's "traits" of interest, but they also require a large bank of questions analyzed with IRT to determine difficulty [44]. CAT-based assessments excel as placement exams for incoming and transfer students [45], and offer more precise and nuanced measurements for multi-institution research studies [43].

6 Strategies For Managing Learning Debt

Learning debt is a challenge to be met at micro- and macro-levels throughout a program of instruction. In this section, we explore evidence-based approaches for addressing learning debt.

6.1 Student-Centric Strategies

Student-centered strategies often focus on encouraging the development of meta-learning skills (i.e. helping students become more effective learners) and more effectively engaging students with learning activities (as an engaged student is more likely to spend the time and effort needed to drive learning).

The first category helps students take charge of their own learning by de-mystifying the learning process and encouraging them to adopt better study practices. The simple expedient of sharing growth/fixed mindset research before tackling a subject requiring accommodation learning results in impressive learning gains as students are more willing to engage in the productive struggle [46]. The National Academies' *How People Learn* [47] provides a rich, academically-focused synthesis of the scientific groundings of learning, while works like Doyle and Zakrajsek's *The New Science of Learning* [48] and Carey's *How We Learn* [49] provide accessible, student-focused explanations and adoptable strategies for improving their learning. Introducing this content to students early, and continually referencing and modeling it throughout their college career, can help encourage better learning practices, combat impostor syndrome, and foster professional identity.

The second category includes well-researched and widely known strategies such as culturally-relevant pedagogy [50], problem-based learning [51], peer mentors [52], flipped classrooms [53], bite-sized videos [54], specification grading [55], peer instruction [56], POGIL [57], and more, and largely focus on making the learning experience more engaging or less anxiety-inducing for students. These strategies are typically adopted at the course level, though some programs may seek to adopt them across multiple courses or even an entire degree [58].

In incorporating student-focused pedagogical strategies into a course, care must be taken to manage the cognitive load we are instilling in students. For example, problem-based learning adds substantial complexity over traditional programming assignments - managing social interactions with peers, investigating a problem domain, formulating plans, and developing solutions. Adequate practice with each element and ongoing supervision and mentoring is vital to successful learning with this approach. Arranging student learning tasks in order of

increasing complexity and using scaffolding techniques to support early learning is of critical importance.

6.2 Mastery Learning

Mastery learning emerges from Bloom's proposition that any student, given sufficient time, could master a subject [13]. This simple idea can be expressed as an equation:

$$degree\,of\,learning = f(\frac{time\,spent}{time\,needed})$$

In traditional instruction, the instructional time is held constant, resulting in varying *degree of learning* across the student population. Mastery learning instead varies the instructional time spent in an effort to ensure **all** students meet the desired *degree of learning*. There are multiple avenues for achieving this increased instruction, from incorporating extra time in the classroom, offering supplemental tutoring and instruction to students who need it, and more recently, leveraging instructional technologies [59]. However, each of these approaches shares the attribute of frequent formative assessments to determine when the desired mastery has been achieved. The literature suggests that well-implemented mastery learning approaches can vastly improve student learning outcomes, but do not map well to the traditional one-size-fits-all college instructional model, presenting significant challenges for adoption [60, 61].

6.3 Spiral Curriculum

Turning to the program/curriculum level, we look to strategies like Bruner's concept of a *spiral curriculum* – one that periodically returns to a topic or skill and both revisits and expands upon it [62]. This strategy helps mitigate learning debt by revisiting and reviewing a foundational skill or concept (offering the student another chance to grapple with it directly) and then explicitly linking that foundational concept to more advanced concepts that build on it. For example, rather than introducing an exhaustive list of HTML tags, a web development course might focus on a handful of most commonly used tags, then shift to using CSS to style them, introduce a handful more HTML tags, and continue interspersing new HTML tags and CSS rules between interactive JavaScript techniques, building towards a full understanding of client-side web technologies. Such approaches are surprisingly effective; for example, Lionelle et al. adopted a spiral curriculum approach in their CS1 course and found it increased student final exam scores by 10%, improved retention into CS2 for women by 19.2% and 9.2% for all students, and produced a 15% increase in passing grades in CS2 [63].

6.4 Individualized Instruction

Catering instruction to students' interests and prior experiences has long been a staple of the progressive education movement, and a natural refutation of the "banking model" of education [64]. However, this is difficult to do at scale, and many of the research-based instructional practices in section 6.1 represent adaptations to quasi-individualize instruction. The introduction of computers in education excited possibilities of truly individualized

instruction at scale, though the promise did not bear out at the time [65, 66]. With the generative AI boom, these hopes have been reignited and may yet bear fruit [67].

7 Conclusions And Future Work

In this paper we have laid the groundwork for recognizing what we term *learning debt*, observed as the effect of competency gaps in cumulative learning, and provided the analogy of technical debt in software engineering which we suspect will resonate with many in the computer science education discipline (as well as our students). It is our hope that this construct may provide a guiding principle for instructional design, student-focused interventions, and formulation of future research questions within the CS education community. With the advent of generative AI, we fear the concept of learning debt will become especially salient (as suggested by the preprint [68]).

We have also laid out many strategies for addressing this learning debt, all with significant evidentiary basis in the literature. We encourage the community to consider wider adoption of research-backed instructional strategies and to embrace the shift from knowledge-oriented to competency-oriented standards, as both of these represent approachable goals that consider learning in a more holistic and student-oriented manner. However, the biggest gains in helping all learners will likely come from developing robust and nuanced competency assessment approaches and reorganizing educational efforts to embrace mastery learning approaches organized using spiraling curriculum structures along with individualization in what Doll refers to as a "post-modern" curriculum [16].

References

[1] S. Frezza, M. Daniels, A. Pears, Å. Cajander, V. Kann, A. Kapoor, R. McDermott, A.-K. Peters, M. Sabin, and C. Wallace, "Modelling competencies for computing education beyond 2020: a research based approach to defining competencies in the computing disciplines," in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, pp. 148–174.

[2] C. T. Force, *Computing Curricula 2020: Paradigms for Global Computing Education*. New York, NY, USA: Association for Computing Machinery, 2020.

[3] A. N. Kumar, R. K. Raj, S. G. Aly, M. D. Anderson, B. A. Becker, R. L. Blumenthal, E. Eaton, S. L. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, and Q. Xiang, *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024.

[4] B. Cambourne, *The whole story: Natural learning and the acquisition of literacy in the classroom*. ERIC, 1988.

[5] N. Chomsky, "Knowledge of language: Its nature, origin, and use," *New York*, 1986.

[6] J. P. Gee, *Literacy and education*, 2nd ed. Routledge, 2015.

[7] ——, *Situated language and learning: A critique of traditional schooling.* routledge, 2012.

[8] P. Freire, "Pedagogy of the oppressed (30th anniv. ed.)," *New York: Continuum*, vol. 35, 2000.

[9] A. Luke, "On explicit and direct instruction," *Australian Educator*, pp. 32–35, October 2013.

[10] J. P. Gee, "A sociocultural perspective on opportunity to learn," *Assessment, equity, and opportunity to learn*, pp. 76–108, 2008.

[11] R. M. Gagne, "Presidential address of division 15 learning hierarchies," *Educational psychologist*, vol. 6, no. 1, pp. 1–9, 1968.

[12] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. J. Hill, and D. R. Krathwohl, *Taxonomy of Educational Objectives: The Classification of Educational Goals.* Longmans, 1956.

[13] B. S. Bloom, "Learning for mastery. instruction and curriculum. regional education laboratory for the carolinas and virginia, topical papers and reprints, number 1." *Evaluation comment*, vol. 1, no. 2, p. n2, 1968.

[14] A. C. for Computing Education in Community Colleges (CCECC), *Bloom's for Computing: Enhancing Bloom's Revised Taxonomy with Verbs for Computing Disciplines.* New York, NY, USA: Association for Computing Machinery, 2023.

[15] J. Piaget and E. Duckworth, "Genetic epistemology," *American Behavioral Scientist*, vol. 13, no. 3, pp. 459–480, 1970.

[16] W. E. Doll Jr, *A post-modern perspective on curriculum.* Teachers College Press, 1993.

[17] L. S. Vygotsky, *Mind in society: The development of higher psychological processes.* Harvard university press, 1978, vol. 86.

[18] J. L. Plass, R. Moreno, and R. Brünken, *Cognitive load theory.* Cambridge university press, 2010.

[19] F. Paas, A. Renkl, and J. Sweller, "Cognitive load theory and instructional design: Recent developments," *Educational psychologist*, vol. 38, no. 1, pp. 1–4, 2003.

[20] J. Piaget, *Piaget's theory of intelligence.* Englewood Cliffs, NJ: Prentice Hall, 1978, vol. 2.

[21] R. Lister, "Toward a developmental epistemology of computer programming," in *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, ser. WiPSCE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 5–16. doi: 10.1145/2978249.2978251 . [Online]. Available: https://doi.org/10.1145/2978249.2978251

[22] D. Teague, "Neo-piagetian theory and the novice programmer," Ph.D. dissertation, Queensland University of Technology, 2015.

[23] Institute of Educational Sciences, "Data point: Adult literacy in the united states." [Online]. Available: https://nces.ed.gov/pubs2019/2019179/index.asp#:~:text=Four%20in%20five%20U.S.%20adults,nativity%20status:%202012%20and%202014

[24] B. Xie, M. J. Davidson, M. Li, and A. J. Ko, "An item response theory evaluation of a language-independent cs1 knowledge assessment," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 699–705. doi: 10.1145/3287324.3287370 . [Online]. Available: https://doi.org/10.1145/3287324.3287370

[25] M. J. Davidson, B. Wortzman, A. J. Ko, and M. Li, "Investigating item bias in a cs1 exam with differential item functioning," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1142–1148. doi: 10.1145/3408877.3432397 .

[26] M. Ali, P. Rao, Y. Mai, and B. Xie, "Using benchmarking infrastructure to evaluate llm performance on cs concept inventories: Challenges, opportunities, and critiques," in *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, ser. ICER '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 452–468. doi: 10.1145/3632620.3671097 .

[27] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, validation, and use of a language independent cs1 knowledge assessment," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ser. ICER '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 93–101. doi: 10.1145/2960310.2960316 . [Online]. Available: https://doi.org/10.1145/2960310.2960316

[28] 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*, 2015, pp. 2–10.

[29] Z. Wu and D. H. Smith, "Evaluating micro parsons problems as exam questions," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 674–680. doi: 10.1145/3649217.3653583 . [Online]. Available: https://doi.org/10.1145/3649217.3653583

[30] R. P. Y. Lai, "Beyond programming: A computer-based assessment of computational thinking competency," *ACM Trans. Comput. Educ.*, vol. 22, no. 2, Nov. 2021. doi: 10.1145/3486598 . [Online]. Available: https://doi.org/10.1145/3486598

[31] M. Berges and P. Hubwieser, "Evaluation of source code with item response theory," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 51–56. doi: 10.1145/2729094.2742619 .

[32] C. S. Dweck and A. Master, "Self-theories and motivation: Students' beliefs about intelligence," in *Handbook of motivation at school*. Routledge, 2009, pp. 137–154.

[33] B. Rammstedt, D. J. Grüning, and C. M. Lechner, "Measuring growth mindset," *European Journal of Psychological Assessment*, 2022.

[34] B. Midkiff, M. Langer, C. Demetriou, and A. Panter, "An irt analysis of the growth mindset scale," in *Quantitative Psychology: The 82nd Annual Meeting of the Psychometric Society, Zurich, Switzerland, 2017*. Springer, 2018, pp. 163–174.

[35] S. M. Werner and Y. Chen, "Evaluating identity and belonging in computer science students: Instrument adaptation and analysis," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 1421–1427. doi: 10.1145/3626252.3630840 . [Online]. Available: https://doi.org/10.1145/3626252.3630840

[36] L. Östlund, N. Wicklund, and R. Glassey, "It's never too early to learn about code quality: A longitudinal study of code quality in first-year computer science students," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 792–798. doi: 10.1145/3545945.3569829 . [Online]. Available: https://doi.org/10.1145/3545945.3569829

[37] C. Izu and C. Mirolo, "Exploring cs1 student's notions of code quality," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 12–18. doi: 10.1145/3587102.3588808 . [Online]. Available: https://doi.org/10.1145/3587102.3588808

[38] N. H. Bean, *Algorithmic pedagogy: Using code analysis to deliver targeted supplemental instruction*. Kansas State University, 2022.

[39] J. Edwards, K. Hart, and C. Warren, "A practical model of student engagement while programming," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, ser. SIGCSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 558–564. doi: 10.1145/3478431.3499325 .

[40] J. Leinonen, K. Longi, A. Klami, and A. Vihavainen, "Automatic inference of programming performance and experience from typing patterns," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 132–137. doi: 10.1145/2839509.2844612 . [Online]. Available: https://doi.org/10.1145/2839509.2844612

[41] F. E. James, "Advancing computer science education: Strategies for student support and teacher development," Ph.D. dissertation, Kansas State University, 2025.

[42] Z. Gao, C. Cui, H. Yan, J. Liu, X. Sun, and J. Feng, "Towards a quantitative competency model for cs1 via five-channel learning sequences," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, 2025, p. 367–373. doi: 10.1145/3641554.3701837 .

[43] D. J. Weiss and G. G. Kingsbury, "Application of computerized adaptive testing to educational problems," *Journal of educational measurement*, vol. 21, no. 4, pp. 361–375, 1984.

[44] D. J. Weiss and A. Sahin, *Computerized adaptive testing: From concept to implementation*. Guilford Publications, 2024.

[45] R. J. Gordon, "Using computer adaptive testing and multiple measures to ensure that students are placed in courses appropriate for their skills." 1999.

[46] C. S. Dweck, *Mindset: The new psychology of success*. Random house, 2006.

[47] J. D. Bransford, A. L. Brown, R. R. Cocking *et al.*, *How people learn*. Washington, DC: National academy press, 2000, vol. 11.

[48] T. Doyle and T. Zakrajsek, "The new science of learning," *How to Learn in Harmony with Your Brain: Stylus Publishing*, 2013.

[49] B. Carey, *How we learn: The surprising truth about when, where, and why it happens*. Random House Trade Paperbacks, 2015.

[50] B. Hall, S. Grantham, and R. Whyte, "Embedding culturally relevant pedagogy in practice: Considerations for training and resource development," in *Proceedings of the 8th Conference on Computing Education Practice*, ser. CEP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 29–32. doi: 10.1145/3633053.3633058. [Online]. Available: https://doi-org.er.lib.k-state.edu/10.1145/3633053.3633058

[51] A. Ellis, L. Carswell, A. Bernat, D. Deveaux, P. Frison, V. Meisalo, J. Meyer, U. Nulden, J. Rugelj, and J. Tarhio, "Resources, tools, and techniques for problem based learning in computing," in *Working Group Reports of the 3rd Annual SIGCSE/SIGCUE ITiCSE Conference on Integrating Technology into Computer Science Education*, ser. ITiCSE-WGR '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 41–56. doi: 10.1145/316572.358296.

[52] E. Melville, M. A. Wright, J. Rosales, S. Akhtar, and R. N. Wright, "Improving undergraduate computing engagement with computing fellows across disciplines," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, 2025, p. 763–769. doi: 10.1145/3641554.3701842. [Online]. Available: https://doi-org.er.lib.k-state.edu/10.1145/3641554.3701842

[53] J. Bishop and M. A. Verleger, "The flipped classroom: A survey of the research," in *2013 ASEE annual conference & exposition*, 2013, pp. 23–1200.

[54] A. Manasrah, M. Masoud, and Y. Jaradat, "Short videos, or long videos? a study on the ideal video length in online learning," in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 366–370. doi: 10.1109/ICIT52682.2021.9491115.

[55] B. Harrington, A. Galal, R. Nalluri, F. Nasiha, and A. Vadarevu, "Specifications and contract grading in computer science education," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024.

New York, NY, USA: Association for Computing Machinery, 2024, p. 477–483. doi: 10.1145/3626252.3630929 .

[56] D. Zingaro and L. Porter, "Peer instruction: a link to the exam," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ser. ITiCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 255–260. doi: 10.1145/2591708.2591711 .

[57] H. H. Hu, A. Yadav, D. M. Gavin, C. Kussmaul, and C. Mayfield, "Teamwork in cs1: Student learning and experience with pogil," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 729–735. doi: 10.1145/3545945.3569813 .

[58] S. Laursen, *Levers for Change: An Assessment of Progress on Changing STEM Instruction*, D. Smith, Ed. American Association for the Advancement of Science, 2019. [Online]. Available: https://www.aaas-iuse.org/resource/levers-for-change-an-assessment-of-progress-on-changing-stem-instruction/

[59] C. Szabo, M. C. Parker, M. Friend, J. Jeuring, T. Kohn, L. Malmi, and J. Sheard, "Models of mastery learning for computing education," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, 2025, p. 1092–1098. doi: 10.1145/3641554.3701868 . [Online]. Available: https://doi.org/10.1145/3641554.3701868

[60] C.-L. C. Kulik, J. A. Kulik, and R. L. Bangert-Drowns, "Effectiveness of mastery learning programs: A meta-analysis," *Review of educational research*, vol. 60, no. 2, pp. 265–299, 1990.

[61] S. Ritter, M. Yudelson, S. E. Fancsali, and S. R. Berman, "How mastery learning works at scale," in *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, 2016, pp. 71–79.

[62] J. S. Bruner, *The Process of Education*. Harvard University Press, 1977.

[63] A. Lionelle, J. Grinslad, and J. R. Beveridge, "Cs 0: Culture and coding," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 227–233.

[64] J. Dewey, "Experience and education," in *The educational forum*, vol. 50, no. 3. Taylor & Francis, 1986, pp. 241–252.

[65] D. Deep, "The computer can help individualize instruction," *The Elementary School Journal*, vol. 70, no. 7, pp. 351–358, 1970.

[66] J. E. Coulson, "Computer-assisted instruction and its potential for individualizing instruction." 1970.

[67] E. Logacheva, A. Hellas, J. Prather, S. Sarsa, and J. Leinonen, "Evaluating contextually personalized programming exercises created with generative ai," in *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1*, ser. ICER '24. New York, NY, USA: Association for Computing

Machinery, 2024, p. 95–113. doi: 10.1145/3632620.3671103 . [Online]. Available: https://doi-org.er.lib.k-state.edu/10.1145/3632620.3671103

[68] N. Kosmyna, E. Hauptmann, Y. T. Yuan, J. Situ, X.-H. Liao, A. V. Beresnitzky, I. Braunstein, and P. Maes, "Your brain on chatgpt: Accumulation of cognitive debt when using an ai assistant for essay writing task," 2025. [Online]. Available: https://arxiv.org/abs/2506.08872