Computing-specific pedagogies and theoretical models: common uses and relationships

LAURI MALMI, Aalto University, Finland
JUDY SHEARD, Monash University, Australia
CLAUDIA SZABO, The University of Adelaide, Australia
PÄIVI KINNUNEN, University of Helsinki, Finland

Computing education widely applies general learning theories and pedagogical practices. However, computing also includes specific disciplinary knowledge and skills, e.g., programming and software development methods, for which there has been a long history of development and application of specific pedagogical practices. In recent years, there has also been substantial interest in developing computing-specific theoretical models, which seek to describe and explain the complex interactions within teaching and learning computing in various contexts. In this paper, we explore connections between computing-specific pedagogies and theoretical models as reported in the literature. Our goal is to enrich computing education research and practice by illustrating how explicit use of field-specific theories and pedagogies can further the whole field. We have collected a list of computing-specific pedagogical practices and theoretical models from a literature search, identifying source papers where they have been first introduced or well described. We then searched for papers in the ACM digital library that cite source papers from each list, and analyzed the type of interaction between the model and pedagogy in each paper. We developed a categorization of how theoretical models and pedagogies have supported or discounted each other, have been used together in empirical studies or used to build new artefacts. Our results showed that pair programming and parsons problems have had the most interactions with theoretical models in the explored papers, and we present findings of the analysis of these interactions.

CCS Concepts: • Social and professional topics → Computing education.

Additional Key Words and Phrases: computing education, theory, pedagogy, pair programming, Parsons problems

ACM Reference Format:

1 INTRODUCTION

Computing Education Research (CER) is a relatively young, evolving field of science. Recent research has highlighted the ways the field has developed [5, 83, 89], with further studies focusing on new computing education-specific theories [53–55]. The emergence of field-specific theories and models is regarded as a sign of a maturing research field [31, Chapter 1]. This paper positions itself into this same line of research by investigating the connections between

Authors' addresses: Lauri Malmi, Aalto University, Finland, lauri.malmi@aalto.fi; Judy Sheard, Monash University, Australia, judy.sheard@monash.edu; Claudia Szabo, The University of Adelaide, Australia, claudia.szabo@adelaide.edu.au; Päivi Kinnunen, University of Helsinki, Finland, paivi.kinnunen@helsinki.fi.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2024 Association for Computing Machinery.

Manuscript submitted to ACM

the emerging computing-specific theoretical aspects of learning and pedagogical practices in computing classrooms and other educational contexts. Educational theory seeks to better understand teaching and learning related phenomena to find ways to support students' learning and we argue that such theories should inform development of practice. The amount and depth of inter-connections between computing-specific theories and pedagogies thus highlight the evolution of CER as a research field, specifically whether it has matured sufficiently to produce theories and pedagogical practices that impact the development of each other.

When considering the quality of teaching and learning, the importance of the relationship between learning theory and pedagogical practice is evident. The knowledge of general and field-specific learning theories are a focal part of a teacher's pedagogical content knowledge, which is also grounded in knowledge about how a particular topic could be taught [80]. Learning theories provide teachers with some basic tenets on which to build their pedagogical choices. For instance, pedagogies, such as problem-based learning that require students to work together and question what they already know about the topic and what still needs to be learned in order to complete the task at hand, are well in line with the constructivist learning theory. General learning theories are widely applicable in computing education, and there is a large corpus of research addressing pedagogical practices in computing education that is based on general educational research literature, e.g., [30, 48, 60, 71]. However, the computing discipline also has its own specialized disciplinary knowledge and skills, e.g., programming, which are unique and may be best taught with computing-specific pedagogical practices, informed also by computing-specific theoretical models. The complete mechanisms of pedagogical influence are currently far from being fully understood, and much more theoretical and empirical work is needed; indeed, educational settings are so diverse and complex that building accurate predictive theories is a rare option, if possible at all. Tedre and Pajunen [88] therefore argue for the development of models in educational settings ¹. Theoretical models, which identify relevant factors and their interaction can, however, be very useful to better understand the complexities of teaching and learning processes and support systematic development of teaching. In this respect, the results of this study aim to contribute to our collective pedagogical content knowledge by bringing forward the interplay of emerging field-specific theories and pedagogies of learning computing. We are not aware of any survey of such theory-pedagogy relationships focusing on computing education.

Several large scale reviews have identified substantial research on building theoretical models of teaching and learning computing [47, 53–55]. Malmi et al. also explored whether computing-specific theories had informed teaching practice in computing education finding some limited evidence of their impact [57]. However, their point of view focused on whether a theory was "Used to design a new pedagogical method" [Ibid, p.5]. In this paper, we seek to look at the theory-pedagogy interaction from a wider perspective. Theories can also provide support for teachers' pedagogical choices by giving insight into how the pedagogy influences the learning process and theories can systematically further the development of pedagogies by giving arguments as to why an existing pedagogy is successful or not. On the other hand, a pedagogy can support a theory by giving evidence of whether the specific implementation of the learning process matches with the theoretical explanation. Moreover, both theory and pedagogy can be used to discuss and analyze findings in a complex situation. Thus, we set our goal to explore this interaction much more broadly than has been carried out in earlier research [57]. Our focal interest in computing-specific theories and pedagogies stems from our view that the field – computing education research (CER) – should develop its own deep understanding of how teaching and learning of computing takes place, what factors are involved and how practice is informed of their role.

¹They discuss in depth the complexity of interpreting the concept "theory". Such discussion is, however, beyond the scope of this paper, and we therefore use terms "theory", "model" or "theoretical model" without denoting any specific difference between them.

2 COMPUTING-EDUCATION-SPECIFIC THEORIES AND PEDAGOGIES

Each discipline has its own characteristics and challenges for learning, and computing, with its roots in three different research traditions [90], is no different. First, the *mathematics* tradition emphasizes formal presentation, coherent theoretical structures, and creation and proof of hypotheses and theorems. Much research in theoretical computer science, algorithms, and machine learning follows this tradition. Second, a significant part of computing follows the *scientific* tradition, which builds on forming hypotheses, constructing models and making predictions based on these, designing experiments, and collecting data and analyzing results, in an iterative approach until the model is sufficiently accurate. [19]. Much of computing education research builds on this tradition. The third tradition in computing follows an *engineering* approach. The construction of software and hardware artifacts has always been a core computing activity, with design as its foundation block, as Denning et al. [19, p.64] formulated: "Design is the bedrock of engineering: engineers share the notion that progress is achieved primarily by posing problems and systematically following the design process to construct systems that solve them."

This richness of the computing tradition poses significant challenges for computing educators as very different pedagogical methods are needed in various courses, e.g., formal mathematical analysis, theorem development and proving, as opposed to software development in large-scale projects. Furthermore, many, perhaps most, core computing concepts are abstract in the sense that there is no obvious corresponding "real world" physical object that would be familiar to students in their everyday life. Teachers can build some of these connections between "abstract" and "real", as is carried out, for example, in CS unplugged pedagogy [7] and tangible computing [39]. Finally, learning computing is not just about learning concepts and their relationships, but also involves learning many abstract skills, such as conceptual analysis, problem solving, programming, testing, debugging, software design. It is thus apparent that computing-specific pedagogical methods are needed to complement generic pedagogical methods.

Shulman [81] presents the concept of *signature pedagogy*, which denotes pedagogical practices that support the preparation of students for a specific profession, for example, bedside teaching at medical schools or case dialogues at law schools. In a signature pedagogy the goal is to learn about ways of thinking within the profession beyond working practices. Shulman distinguishes three levels of a signature pedagogy. First, there is *surface structure*, i.e., concrete, operational acts of teaching and learning, such as showing, demonstrating, questioning, interacting, etc. Second, there is *deep structure*: "assumptions about how best to impart a certain body of knowledge and know-how", and finally there is *implicit structure*, "a moral dimension that comprises a set of beliefs about professional attitudes, values, and dispositions." [81, p55].

When considering computing education, an example of a signature pedagogy could be pair programming, where students familiarize themselves with programmers' working methods: working in pairs and alternating between the driver and navigator roles (surface structure) during a coding session. While doing this, they learn to read, write, discuss and critique code (deep structure) and learn about professional values: code quality matters (implicit structure). On the other hand, not all computing-specific pedagogical practices are signature pedagogies. for example, Parsons problems [67], is a technique which supports learning to code by simplifying the coding task into building a puzzle of program statements instead of writing all the code. While this method has demonstrated positive impact on learning [29], it is not a professional practice.

Falkner and Sheard [30] present a comprehensive overview of pedagogical approaches in computing education. They identify and discuss six *pedagogical approaches*, i.e., broader guidelines to implement teaching, each of which may cover a number of specific *pedagogical practices*, denoting a specific activity within a course. These are *Active learning*,

Collaborative learning, Cooperative learning, Contributing student pedagogy, and Blended learning and MOOC. Many of the example practices are generic, as one would expect, such as flipped classroom as an activity implementing blended learning, or content creation as an activity implementing a contributing student pedagogy. However, they also present several examples of computing-specific activities, such as, live coding or test-driven development as instances of active learning or pair programming as an instance of cooperative learning.

Sanders et al. [76] explored the concept of *active learning* by surveying the computing education community as well as carrying out a systematic literature review to find out how active learning is interpreted and what kind of practices are associated with it. Their results implied that while active learning is widely considered a good thing, it is often not a well defined approach, and it is rarely connected to learning theories. They identified 38 different activities that implement active learning in some way, and classified them into five broader categories: Lecture content outside the lecture, Activities during lecture time, Collaboration and social engagement, Techniques from other disciplines, and Software development techniques. They also identified a number of techniques focused on the change in the instructor's perspective of teaching rather than specific student activities.

The use of theories in computing education research has received substantial attention in recent years, with several reviews and special issues focusing on this theme. Malmi et al. [56] analyzed papers published in the ICER conference, and two major journals, ACM Transactions on Computing Education (TOCE) and Computer Science Education (CSE) from 2005-2011, identifying the use of a large number of theories, models and frameworks; although, these constructs were only found in only roughly half of the papers. They also analyzed the origin discipline of the constructs, where the largest groups had their origin in educational sciences, psychology and computing. A small set of constructs had been developed in computing education research. Lishinski et al. [47] complemented this analysis by looking at papers published in ICER and CSE in 2012-2015 identifying significantly higher number of papers using theory outside CER than in the earlier review [56]. A few years later, Szabo et al. [86] carried out an extensive review of the use learning theories in the CER literature. They collected a large set of general learning theories from [2] augmenting it with theories from [53, 56] and searched ACM Digital Library for their occurrences in papers published in different venues. They closely analyzed the key actors in a set of theories, how they interacted and what internal aspects (cognitive, affective, behavioral) were involved. A further analysis investigated how theories were connected with each other by determining where two theories occurred in the same publication and using this to identify three clusters of theories that worked together: experiental theories, theories of mind. and social theories. This work was continued by Szabo and Sheard [87], who investigated more closely the interaction of theories. Looking at papers that cited two theories, they presented a more detailed clustering of learning theory communities: behaviourist and cognitivist learning theories, working memory and experiential learning theories, motivation and behaviourism learning theories, as well as a computing education learning theories community. They investigated the computing education learning theories community more closely, by categorizing the type of interactions the pairs of theories had in the papers. The categories ranged from cases with just casual mentions to cases where theories were discussed separately or together or they were critically compared, and further to cases when both theories were used to analyze results or inform some design or finally the build or extend some artefact or theory.

Computing-specific learning theories or models have also been explored Malmi et al. (calling them domain-specific theories) in multiple papers [53–56]. They identified in total 124 theoretical constructs (including theories, models, frameworks and theory-based instruments) in eleven different focus areas in computing education research: assessment/self-assessment, computing education research, content/curriculum/learning goals, emotions/attitudes/beliefs/self-efficacy, errors/misconceptions, learning/understanding, learning behavior/strategies, perceptions of computer science/computing, Manuscript submitted to ACM

performance/progression/ retention, study choice/orientation and teaching/pedagogical content knowledge ². They analyzed the methods used to develop the constructs, for what purposes they had been developed, and how they had been used in further research, and citing the original papers where the constructs had been presented. In their later work, they analyzed whether these computing-specific theoretical constructs had had any impact on developing or suggesting new pedagogical practices, and found very little evidence of this. [57].

3 RESEARCH QUESTIONS AND METHOD

In this work, we seek to explore more widely and deeply the connections between computing-specific pedagogies and theoretical developments. Malmi et al. [57] explored whether computing-specific theories and models had informed new pedagogical developments, which were presented either in papers that cite the theory paper or as pedagogical implications in the discussion part of the theory paper itself. However, theories and pedagogies can interact in many different ways, e.g., as shown by Szabo and Sheard [87] between different computing-specific theories.

To better understand relationships between computing-specific pedagogies and theoretical developments, and given the lack, to the best of our knowledge, of any broad analysis of computing-specific pedagogies, our first goal is to identify a set of such pedagogies, as reported in the CER literature, followed by an identification of potential locations of theory-pedagogy relationships and an analysis of relationship depth. Our research questions are:

- (1) RQ1: What computing-specific pedagogies are defined in the CER literature?
- (2) RQ2: What computing-specific learning theories and models ³ identified by a name are discussed in the CER literature?
- (3) RQ3: What kind of connections have been made between computing-specific theories/models and pedagogies in CER?

To better understand the results to the third research question, we focused on the cases with many connections between theories and a specific pedagogy. The two pedagogies with the most connections were pair programming and Parsons problems, and we therefore added two subquestions.

- (1) RQ3.1: What type of connections have been made between computing-specific theories and pair programming?
- (2) RQ3.2: What type of connections have been made between computing-specific theories and Parsons problems? Data was collected and analyzed in several phases, as discussed below.

3.1 Identifying computing-specific pedagogies

Pedagogy is a concept that has no clearly agreed definition. In general, it can be interpreted as a methodological approach, technique or activity that teachers apply to support their students to achieve certain learning goals or other objectives, such as increasing motivation or engagement. A pedagogy can cover a holistic process, which integrates several techniques over a longer period. For example, in project-based learning, students can work in teams to address a complex open problem during a whole semester using several practices, such as team discussions, self-studying, following supplementary lectures, and designing and/or building some artefacts and writing reports. On the other hand, a pedagogy can cover specific techniques applied in a classroom for a limited time, such as analyzing fading worked examples given by a teacher for a specific topic.

 $^{^2}$ See [55] for their definition of these areas.

³As mentioned earlier, in this paper the distinction between the nature of theories and models is not relevant to the discussion. We use the term theory for denoting both of them.

Numerous pedagogical methods are generic in the sense that they are applicable to many different disciplinary contexts, including computing education. We are, however, interested in *computing-specific pedagogies*, i.e., methods which have been developed in computing education contexts and can be applied (almost) only there. Examples of such methods include pair programming [36, 37], Parsons problems [21, 25], computer science unplugged [1, 7] or Use-Modify-Create [52]. We are not aware of any comprehensive list of computing-specific pedagogies. Falkner and Sheard [30] presented a deep discussion on a number of pedagogical approaches; however, they did not build a comprehensive list. Sanders et al. [76] discussed active learning approaches listing many specific pedagogies, supporting these in some way. However, the computing education education literature is vast, presenting a multitude of reported practices that are often tied to a certain context, depending on the teacher's experience, available teaching and learning resources (both hardware and software as well as physical environment), schedule and target student group. We acknowledge that building a comprehensive list of computing-specific pedagogies is likely not possible due to the scope of literature published in over 50 years. However, it is feasible to build a representative list which we set as our goal. We therefore proceeded, as follows.

- (1) We collected pedagogies discussed in the previously mentioned papers [30, 76].
- (2) We searched systematically through five years of papers (2018-2022) published in the ITICSE, SIGCSE, and ICER conferences as well as in ACM TOCE and CSE journals, to identify any new computing-specific pedagogical practices and techniques. We read through the title, abstract, and keywords to identify candidates, and then the full paper to describe them. Each paper was analyzed by one author, who identified relevant candidates for such pedagogies. The candidates were discussed jointly to decide whether they could be applicable in a variety of computing education contexts. The criteria for ruling out a pedagogy were if the pedagogy required use of some specific software, which had been developed for local needs and was likely not accessible by others, or if the pedagogy was tailored for some special group of students, for example, based on their cultural knowledge and background. We also excluded pure professional practices that are used in education, such as test-driven development that can be considered more as learning goals of professional skills than pedagogical practices. We also excluded papers focusing on micro level pedagogies, such as a single or set of assignments.
- (3) The summary list was augmented with expert knowledge of computing-specific pedagogies by three authors, all of whom had 10-30+ years of experience in teaching computing courses as well as an almost equally long experience in publishing in computing education research venues and who were thus highly knowledgeable of various pedagogical approaches that had been used and reported in literature.
- (4) The final list covered 23 pedagogies, which are listed in Appendix A. For each of these, we tried to find a paper where the pedagogy was first published. However, this was not successful in all cases, and so we then identified an early paper that was widely cited in Google Scholar. We call these papers *pedagogy source papers*.

3.2 Identifying computing-specific theories

Our next goal was to identify a list of computing-specific theories in computing education. Here we built on the work by Malmi et al. who had explored these in several papers [53–55] identifying well over 100 constructs in papers published in ICER, TOCE and CSE during 2005-2020, which they called *theoretical constructs*. This list was augmented by the work by Szabo and Sheard [87] who had also analyzed the use of learning theories in computing education. However, they used a broader search identifying additional computing-specific theoretical constructs. When reviewing this list, we observed that many of the identified constructs reported in [53–55] were early developments, such as statistical Manuscript submitted to ACM

models, grounded theories or phenomenographical outcome spaces, while the list in [87] had theories with more broadly recognized names, such as *engagement taxonomy* [64] or *learning edge momentum* [70]. Considering our main goal of finding connections between computing education specific theories and pedagogies, we decided to focus on more established theories or models for which we could identify a name, either proposed by the original authors or used in other citing papers later on.

In order to test whether this approach seemed promising, we used these names as keywords, searching them from the Scopus data base for their occurrence in titles, author keywords or abstracts. It seemed likely that if they were recognized with a name, we would find hits for them in the data base. Indeed, for most of them, we found many hits, while some had none. We therefore decided to focus on a list of 21 theories for which we found hits in Scopus, and use these in further analysis. These theories and models are listed in Appendix B. For each of them, we identified the original paper where it was published, as they were reported in [53–55, 87]. We refer to these papers as *theory source papers*.

3.3 Identifying connections of theories and pedagogies

In order to identify connections between computing education specific theories and pedagogies, we used the lists of pedagogy and theories source papers, as follows. First, we identified all papers in the ACM Digital Library that cite a theory source paper, and correspondingly a pedagogy source paper. Finally, we cross-tabulated these papers with a script to identify all papers that cite one or more papers in both lists. We refer to these papers as *intersection papers*.

A paper may cite both a theory source paper and a pedagogy source paper for multiple reasons. We therefore developed a categorization scheme that would describe various ways a theory and pedagogy could be discussed and possibly connected in the paper. The initial categorization was revised several times when we analyzed the intersection papers, finally leading to the following categories (Table 1).

Category	Description	Level
Separate discussion	Theory and pedagogy are casually referenced in the paper or list of references, or possibly discussed in related work but there is no connection to the empirical study in the paper.	1
Discussed in context without relationship	Theory and pedagogy are both discussed in relation to the empirical work reported in the paper, but no relationship between them is mentioned.	2
Explicitly connected in context	The theory and pedagogy are explicitly connected in the context of the reported empirical work.	3
Analysis	Theory and pedagogy are both used in the analysis or discussion of results. Theory is used to explain results of a pedagogy.	4
Theory development	Pedagogy is used to develop, support, or discount an existing theory.	5
Pedagogy development	Theory is used to develop, support, or discount an existing pedagogy.	6
Artefact development	Theory and pedagogy are both used to design or develop another new theory/model/framework/instrument/pedagogy.	7

Table 1. Theory-Pedagogy Interaction Categories

Note for categories 1-4. A vast majority of the analyzed papers presented empirical work, and we investigated whether a theory or pedagogy had some relation to it. However, a few papers were either review papers or papers focusing on theoretical development/discussion. For these cases, we investigated whether the theory or pedagogy was explicitly

discussed and possibly used in argumentation in the review or theoretical discussion. For brevity, we do not separately discuss these cases from empirical papers when reporting the results.

To analyze the interaction papers, all four authors read each paper in the set of papers from a jointly agreed theory-pedagogy combination, classifying each paper according to one of the categories in the Theory-Pedagogy Interaction Categories scheme. Thereafter, each paper was jointly discussed by all authors until a consensus of the categorization was agreed upon. In cases where the categorization scheme was refined, the previously analyzed papers were revisited to match the revised categories.

After this analysis, as a further verification, we sorted the analyzed papers based on their categorizations. Thereafter all papers in categories 3-7 were revisited by all four authors to confirm that the papers in each category were aligned with the category definition. In this process, a few papers were re-categorized and discussed until a final joint agreement was reached.

4 RESULTS

We present our results below as relevant to each research question.

4.1 RQ1: What computing-specific pedagogies are defined in CER literature?

Our data collection of computing-specific pedagogies from a systematic search of five years of publications (2018-2022) in five venues (ITICSE, SIGCSE, ICER, TOCE and CSE), a search of two key publications [30, 76], and expert knowledge found over 100 pedagogical models and techniques. This list was refined to remove pedagogies that were deemed too specific, were reliant on a particular technology or were considered just training professional practices. The final list of 23 pedagogical models and techniques is presented in Appendix A.

Most of the pedagogies we found are focused on models and techniques for teaching (20) with just three focused on assessment (executable exams, in-flow peer review, peer code review). The pedagogies range from broad models of how a computing course would be taught (e.g., bootcamp and hackathon) or an underlying theme or emphasis in a course (e.g., socially-responsible computing and computing for the social good), to specific techniques used to teach a particular topic or skill (e.g., pair programming and plans and goals).

The most common pedagogies we found are concerned with learning or assessment of programming (15). Two are focused on teaching fundamental computing concepts (*computer science unplugged* and *notional machine*) with two others focused on more advanced topics (*scrumage* and *structured mentorship*). A couple of recently developed pedagogies are specific to cybersecurity (*adversarial mindset in cybersecurity* and *cryptographic playground*).

4.2 RQ2: What computing-specific learning theories and models identified by a name are discussed in CER literature?

We formed a list of computing domain-specific theories from Malmi et al. [53–55] and Szabo and Sheard [87]. The list was refined to include only theories that are identifiable by name. The final list of 21 theories is presented in Appendix B.

Using a classification of theory area of focus developed by Malmi et al. [53] we found that the theories covered 8 of the 11 focus areas defined by Malmi et al. The most common focus area was *learning and understanding* (8), with three others closely connected with learning (*learning behaviour*(4), *performance/progression/retention* (2), and *emotion/beliefs/attitudes/self-efficacy* (2)). The remaining five are spread over the areas of *computing education research* Manuscript submitted to ACM

(2), contents/curriculum/learning goals (1), errors/misconceptions (1), and teaching/pedagogical content knowledge (1). A surprising finding was there were none in the area of assessment/self-assessment.

4.3 RQ3: What kind of connections have been made between domain-specific theories and pedagogies in computing education?

The analysis of intersections of 21 CE theories and 23 CE pedagogies produced 405 intersections. There were five theories that did not intersect with a pedagogy and seven pedagogies that did not intersect with a theory. These are indicated with an asterisk in the lists in Appendixes A and B.

A table showing the frequencies of interactions of the 16 intersecting theories and 16 intersecting pedagogies is shown in Table 4 (Appendix C). We did not include a further 234 cases where *notional machine* as a theory intersected with *notional machine* as a pedagogy. This intersection is shown as an 'x' in Table 4. An investigation of a sample of these cases indicated that there were many papers where the notional machine had been reported as just a theoretical construct or a pedagogy and there was no intersection of theory and pedagogy. We left this analysis for future work.

The most common intersecting theories, producing 60% of all intersections, were notional machine (97), learning edge momentum (76) and theory of instruction for introductory programming skills (70).

The two pedagogies that intersected most with theories were pair programming, which intersected 97 times over 12 different theories and Parsons problems, which intersected 61 times with over 11 different theories. Pair programming and Parsons problems produced 39% of all the intersections. We focused on pair programming and Parsons problems for more detailed analysis.

4.4 RQ3a What type of connections have been made between domain-specific theories and pair programming?

We classified each paper in the set of intersections of pair programming with a theory according to the theory-pedagogy interaction level as described in Table 1. From the 97 papers in the data set, two papers were eliminated as they were duplicates or did not include a reference to the theory or pedagogy in the main text, with the remaining 95 describing intersections between pair programming and theories. A large majority of the intersections fell into categories 1 and 2 as shown in Figure 1, denoting that both the pedagogy and the theory were only casually mentioned in the paper, or discussed separately with no connection to the empirical work in the paper. The frequencies of each category of interaction is shown in Table 2.

We found that pair programming intersected with 12 theories; however, only five theories had intersections at a level of 3 or above. There were no intersections with cognitive complexity of computer programs, normalized program state model (NPSM), progression of early computational thinking model (PECT) or reducing abstraction. We will now describe the cases where we found the strong connections between the pair programming and theory.

4.4.1 Artefact development (Category 7). Pair programming has been strongly connected with several computing-specific theories, namely, abstraction transition taxonomy (ATT), learning edge momentum (LEM), and notional machines (NM), resulting in various artefact developments. ATT represents a foundation theory for PRIMM [79], where three key principles are proposed to guide the teaching of programming, namely, mediation through language, the understanding that learning moves from the social plane to the cognitive plane, and the role of the more knowleadgeble other (MKO) in the Zone of Proximal Development [6]. Mediation through language suggests that students should be encouraged to work together and discuss through a social construction of knowledge. Pair programming is a pedagogical

Table 2. Counts of interactions of Pair Programming and theories at different levels of interaction: 3 Explictly connected in context; 4 Analysis; 5 Theory development; 6 Pedagogy development; 7 Artefact development.

	Ca	ateg	orie	es o	f In	tera	ction
Theory / model name	<3	3	4	5	6	7	Total
Abstraction transition taxonomy (ATT)	8	-	-	-	-	1	9
Engagement taxonomy	8	-	-	-	-	-	8
Error quotient (EQ)	1	1	-	-	-	-	2
Learner-Directed Model	1	-	-	-	-	-	1
Learning edge momentum	21	-	2	-	1	2	26
Notional machine	12	-	-	-	-	2	14
Predict student success (PreSS#)	1	-	-	-	-	-	1
Programming "geek" gene	9	-	-	-	-	-	9
Programming plans	4	-	-	-	-	-	4
Theory of instruction for introductory programming skills	9	-	1	-	-	-	10
Threshold skills	3	-	-	-	-	-	3
Zone of proximal flow	8	-	-	-	-	-	8
Total	85	1	3	-	1	5	95

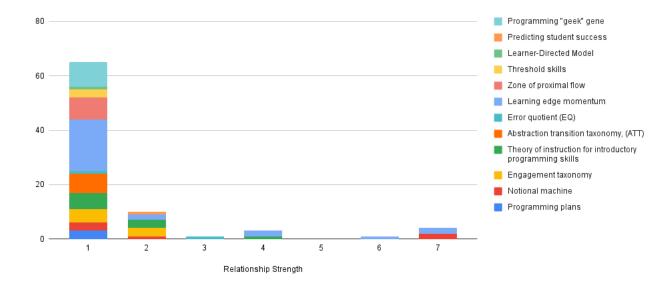


Fig. 1. Connection types between computing-specific theories and Pair programming

technique suggested by the authors as an implementation of this first principle. In [12], pair programming and LEM are used in the development of the pedagogical design for CS1 in a South African university, providing avenues for struggling novice programmers to improve their skills through pair programming. Specifically, as guided by LEM, the proposed approach focused on identifying quickly whether learning was successful, and individual one-on-one support was provided to students who did not succeed in the first weekly assignment. Pair programming was one of the main pedagogies used in the course, together with teaching code reading before writing, and teaching by focusing on time-on-task. Deconstructionism is proposed as a pedagogy in [34], where learners spend as much time deconstructing Manuscript submitted to ACM

code as they do writing code. Notional machines form a backbone for this new pedagogy, and pair programming activities are a type of activity that fits naturally within the proposed pedagogy, among others suggested by the author, such as peer instruction and POGIL.

- 4.4.2 Pedagogy development (Category 6). Wood et al. [95] use LEM to develop best practices for pair programming, specifically by framing the concept of programming confidence, which is defined as the way in which students approach programming exercises and distinguishes between confident students who are willing to experiment with techniques and are unfazed by coding errors, and students who are unable to make independent progress and frequently become 'stuck'. Using LEM and the concept of confidence, authors propose to group students by confidence in their pair programming exercises, and to monitor those students who do not gain confidence very early in the course, in order to provide support.
- 4.4.3 Analysis (Category 4). Analysis relationships were identified between pair programming and LEM and Xie's theory of instruction for introductory programming skills [96]. Both pair programming and LEM are used in the theory-based discussion of feedback in [66]. Hausswolff [92] introduces several theoretical underpinnings based on Dewey [22], Wittgenstein [94] and Deleuze [72] for learning how to program, which are then considered in the analysis of results and experiences of students doing pair programming exercises. The discussion considers both the pair programming undertaken by students under a LEM lens, in particular when attempting to understand the challenges faced by novice programmers. Druga et al. [24] analyse how families designed and developed program games during an in-home study. Xie's theory of instruction for introductory program skills [96] is used to design the analysis codes, and pair programming is recognised as one of the behaviors undertaken by the families building the games, and its dynamic is used to analyse the study results.
- 4.4.4 Explicitly connected in context (Category 3). Lastly, pair programming and learning edge momentum and error quotient theory are explicitly connected in context, respectively, in [43], focused on training teachers in inclusive practice, and in [73], focused on understanding programming behaviors of differently skilled programmers.

4.5 RQ3b: What type of connections have been made between domain-specific theories and Parsons problems?

We classified each paper in the set of intersections of Parsons problems with a theory according to the theory-pedagogy interaction level as described in Table 1. From the 61 papers in the data set, four papers were eliminated as they were duplicates or did not include a reference to the theory or pedagogy in the main text, with the remaining 57 describing intersections between Parsons problems and theories. Similarly to pair programming a large majority of the intersections fell into categories 1 and 2 as shown in Figure 2. The frequencies of each category of interaction is shown in Table 3.

We found that Parsons problems intersected with 11 theories; however, only four theories had intersections at a level of 3 or above. There were no intersections with cognitive complexity of computer programs, learner-directed model, normalized program state model (NPSM), predicting student success, progression of early computational thinking (PECT) model or reducing abstraction. We will now describe the cases where we found the strong connections between Parsons problems and theory.

4.5.1 Artefact development (Category 7). The pedagogy of Parsons problems has been strongly connected within an artefact development relationship with several theories, namely, programming plans [16], notional machine [14, 40]

Manuscript submitted to ACM

Table 3. Counts of interactions of Parsons Problems and theories at different categories of interaction: 3 Explictly connected in context; 4 Analysis; 5 Theory development; 6 Pedagogy development; 7 Artefact development

	C	ateg	orio	es of	f In	tera	ction
Theory / model name	<3	3	4	5	6	7	Total
Abstraction transition taxonomy (ATT)	5	-	-	-	-	-	5
Engagement taxonomy	1	-	-	-	-	1	2
Error quotient (EQ)	3	-	-	-	-	-	3
Learning edge momentum	8	-	-	-	-	-	8
Notional machine	9	1	-	-	-	2	12
Programming "geek" gene	1	-	-	-	-	-	1
Programming plans	2	-	1	1	-	1	5
Theory of instruction for introductory programming skills	16	1	-	-	-	-	17
Threshold skills	1	-	-	-	-	-	1
Zone of proximal flow	3	-	-	-	-	-	3
Total	49	2	1	1	-	4	57

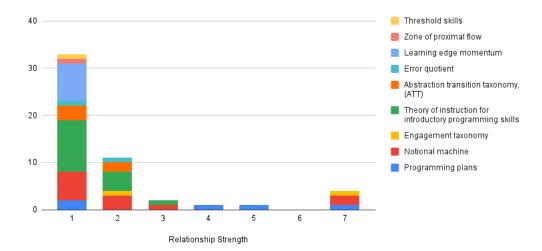


Fig. 2. Connection types between computing-specific theories and Parsons problems

and engagement taxonomy [4]. Cunningham et al. [16] use programming plans and Parsons problems to design a purpose-first programming approach to teach conversational programmers, focusing on learning a handful of domain-specific code patterns and assembling them to create authentic programs. They develop a purpose-first programming prototype that teaches five web scraping patterns. Purpose-first programming enforces a sequence between subgoals driven by variables, where a subgoal uses previously defined variables and/or produces variables to be used in later subgoals. In the development of the artefact, the authors are inspired by Parsons problems devise a three-part activity, in which (i) learners pick from a bank of plan of goals and arrange them in the correct order, (ii) learners repeat the previous activity but with plan code, and (iii) learners fill in the the slots in the code they have assembled. In [40], notional machine and Parsons problems are both used to create new program comprehension activities and to propose new learning trajectories. In [14], a configuration of Snap! is developed using notional machine and Parson problems. Similarly, Parsons problems and engagement taxonomy are used in [4] to create social worked examples, in a reciprocal Manuscript submitted to ACM

connection. Specifically, engagement taxonomy provides guidelines for high engagement, which in turn enhances the effectiveness of Parsons problems.

- 4.5.2 Theory development (Category 5). A theory development relationship was recorded between programming plans and Parsons problems [59]. The authors identify "plan structure errors" when analyzing students' solutions to Parsons problems that are used as a pedagogical activity. Aided by the programming plan error taxonomy [16], the authors define plan structure errors based on what is wrong with the specific placement of the line in the Parsons problem, as plan structure errors when pair of actions animate out of order, control flow error when the line is placed inside the wrong parent control flow block, do both, when an interaction error occurs, do nothing, when the error involves an empty control flow structure that is misplaced relative to another line.
- 4.5.3 Analysis (Category 4). An analysis relationship was found between Parsons problems and notional machine when analysing unplugged activities [62]. Notional machines form a backbone of the unplugged activities under analysis, and programming plans were used to analyse whether the unplugged activities differ from their plugged counterparts, in both computing and mathematics. The use of unplugged activities exposes learners to an accessible and explicit notional machine. Notional machine continues to provide support for programming tasks, that are introduced through a careful progression of activities.
- 4.5.4 Explicitly connected in context (Category 3). Lastly, Parsons problems and the theory of instruction for introductory programming skills [96] were explicitly connected in context in [32], as Parsons problems is a specific exercise type that can be used to explain the relationship between explaining, tracing, and writing skills that are critical within Xie's theory.

5 DISCUSSION

We discuss first our experience in identifying pedagogies and theories and then comment on the results.

5.1 Identifying pedagogies, theories and models

We set our goal to identify a list of computing-specific pedagogies. The task turned out be complex for several reasons. Firstly, pedagogical practices are flexible concepts. They need to be adaptable to different teaching and learning contexts, depending on the specific learning goals, target population as well as teaching and learning resources. In this sense, they cannot too rigidly define the interplay of the relevant factors: activities, students' and teacher(s)' roles and requested resources. Rather, they need to include principles and guidelines how these factors should be used, and teachers will organize the actual implementation of the pedagogy in a particular setting. Therefore, the level of descriptions in publications, how some practice was implemented varied significantly.

Secondly, some pedagogies that have emerged in computing education could also be adapted to other disciplines, such as Structured mentorship, Socially-responsible computing, Computing for social good and In-flow peer review. It is actually plausible that similar type of high-level pedagogies have been developed and used in other disciplines, too. Thus, the borderline between computing-specific and generic pedagogies is not strict. Thirdly, an inherent part of computing education is training some professional practices. For example, if programming courses involve project work where designing programs using UML notation or implementing software using test-driven development is requested, are these pedagogical methods or simply activities to learn a specific professional skill? We deemed the activity itself not a pedagogy in the same way as essay assignment writing is not a pedagogy but just a professional practice. However,

training a skill using pedagogically tailored tasks and actions could be considered a corresponding pedagogy, but such details are often missing from published papers. Fourthly, we deliberately ruled out pedagogical practices which require certain specific software to be used. Computing educators have developed numerous software tools to support learning, e.g.,[51]. However, these tools are very often tailored to match a local context and they may be hard to access. Moreover, they often have little if any support or maintenance for external users [58] making pedagogies built on them vulnerable.

General pedagogies can be applied in multiple disciplinary contexts, while context-specific pedagogies have some components which are (almost) unique for a certain discipline. In computing education, programming is a unique core activity for learning the disciplinary skills.⁴ Moreover, programming is a relevant activity in implementing a large share of advanced computing topics and skills, and it covers multiple computing-specific skills, such as program design, testing, debugging, performance analysis, software architecture design etc. It is thus understandable that most of the computing-specific pedagogies we found are related to teaching programming.

The search for computing-specific theories was more straigthforward, as we could build on existing theory/model lists from literature. Most of the computing-specific theories and models that we deemed more established ones (having a name or an acronym) were related to programming, too.

5.2 Combining theories and pedagogies

As CER is developing and growing as a field of science, it is natural that there is a growing trend of building theories and models which are based on the discipline itself [55]. Thus, we expected to find some interplay of the theories and pedagogical practices. This could happen in many ways. Malmi et al. [57] had explored this interplay from the perspective that theoretical constructs had been used to develop any new pedagogical practices, finding a small number of cases either in papers citing the theory source paper or within the discussion of the source paper itself. They recommended that research papers should more frequently have such discussion of "pedagogical implications" which could make the results more accessible to broader computing educator audience and not only for active computing education researchers. Compared to their goals, our perspective was to explore the interplay of existing pedagogies and theories, which could take place in multiple forms. We indeed found many, over 400 interactions. A detailed analysis of all of them was beyond the scope of this paper and we therefore focused on the most common cases which were related to Pair programming and Parsons problems which we investigated more closely. Other pedagogies with rich interaction with theories were Notional machine (as a pedagogy), Program tracing, Computer science uplugged, Live coding, Media-based computation and Peer code review, all of which are related to learning programming. The most commonly intersected theories were Notional machine (as a theory), Theory of instruction for introductory programming skills, Learning edge momentum, all related to programming, as well.

The closer analysis of pair programming and Parsons problems revealed that a great majority of the intersections were on levels 1 and 2 implying that there was no clear connection between the theory and pedagogy in the paper. This is no surprise, as papers are cited for many reasons, most often simply as related work. The more interesting cases were on levels 3-7, especially on the upper levels. PRIMM pedagogy [79] is a good example. It is explicitly building on the theoretical understanding of how teachers and students discuss in three different levels of language (English, CS Speak, and Code), as the Abstraction transition taxonomy [17] presents. To encourage discussion on multiple levels, which support students to grow into the computing community, the pedagogy integrates heavily pair programming as an activity where students naturally need to discuss much. Clements and Krishnamurthi [14] on the other hand, seeking to

⁴Programming tasks are, of course, used in many other disciplines, too. However, learning programming is part of computing discipline.

Manuscript submitted to ACM

support students to understand notional machine and runtime stacks, were inspired by Parsons problems as a method. They planned a similar tool, implemented with Snap!, where students can work with a interactive visual presentation of stack frames and better understand how they work. These kinds of explicit combinations, where theory provides a suggestion what kind of pedagogical practices would be effective or a pedagogy inspires a new way of learning theory are something which we hope to see more.

Theory and pedagogy can support each other in different ways. Pedagogy can have an impact which matches a theory, thus providing some evidence to support the theory. On the other hand, theory can suggest ways to improve the implementation of an existing pedagogy. Learning edge momentum theory[70] suggests that due to the scaffolded structure of programming knowledge, students who face problems in the beginning of CS1 may increasingly fall behind the others and are in risk of dropping out. Thus, supporting especially weaker students is important. Using best practice from pair programming research, Wood et al. [95] proposed using programming confidence (interpreted as student's current programming skill) as an indicator how to pair students (with similar confidence) as well as switching pairs regularly, finding out that students with low confidence benefited from this practice and increased their confidence.

While our closer analysis focused only on pair programming and Parsons problems, we found many interesting examples of how computing-specific theories/models and pedagogies were used together to further research and improve education. We are confident that analysing the remaining intersection cases would reveal more such cases. This encourages to seek research designs where the pedagogy is explicitly selected or tuned match the theory, or theory is explicitly used to inform pedagogical choices.

5.3 Limitations

Our search for candidate pedagogies covered two key references [30, 76] and five recent years of publications in major computing education research venues, complemented with expert knowledge from three authors with a long experience in computing education and CER. For each of the candidates in the search we discussed it with at least three authors to find consensus whether it could be accepted based on the above criteria. We acknowledge that even in this way, the inclusion/exclusion decision was not clear cut. Therefore we considered the consensus method useful.

We acknowledge that broader search of years and venues would have increased the number of identified pedagogical practices. However, this was beyond the scope of our resources, as it would cover thousands of papers. Moreover, not all well-functioning pedagogies are reported in published papers, and talented teachers have certainly invented many practices which work well in their contexts. However, there is no obvious way to collect such pedagogical content knowledge in a large scale. Surveys to mailing lists and interviews with colleagues would produce some results, but likely many findings would anyway overlap with what we identified in our literature search.

Despite these challenges, we consider building a list of computing-specific pedagogies a meaningful effort, which can be helpful to many readers as a source of educational resources. We acknowledge that such a list is never complete, as the field evolves constantly. For example, the fast developing research related how large language models can be used to support education will certainly create new pedagogical methods, such as prompt problems to guide AI tools to build programs [20].

The decision to focus on theories with some name or acronym was supported by our trial search in Scopus. We considered it plausible that if a theory/model had some clear role in a paper, it would likely be mentioned in the title, abstract or author keywords. We acknowledge that this does not apply to all papers which cite a theory source paper. However, Malmi et al. [55] found that a vast majority of papers citing a theory source paper does not use the construct;

the paper is likely cited for some other reason. We considered that based on this finding, we likely would not miss many cases relevant to our goals.

The intersection search for pedagogies and theories was carried out in ACM digital library only, which — to our conception — covers significant share of international computing education research. Searches for other data bases were therefore excluded to reduce the total workload. Finally, the categorization of theory-pedagogy intersections were carefully carried out by all 4 authors until a consensus of the category for each paper was reached. Thereafter we carried out a second pass comparing papers within a single category with each other until the final consensus for all papers on levels 3-7 were reached. We left out closer analysis of levels 1 and 2 papers, because they were considered uninteresting from the purpose of this work.

6 CONCLUSION

Computing education has developed its own pedagogical practices for decades. Theoretical development is considerably more recent activity and there is growing interest in it [55]. We have explored the interplay of these field-specific pedagogies and theories/models in CER literature identifying many interesting examples where they support each other in different ways. We therefore recommend further work to combine them. Field-specific theories and models can provide valuable arguments for developing new pedagogical practices or further development of existing ones. Instead of building on implicit assumptions, they can provide arguments which are based on empirical results and thus support systematic development of computing education.

In this paper, we have analyzed only intersections of computing-specific pedagogies and theories, as we believe that their combinations are essential to further computing education research and practice. We continue the detailed analysis of theory-pedagogy intersections for other pedagogical practices than pair programming and Parsons problems, which were now left out from this paper. However, there is also much space in future work to explore similar connections between computing-specific and generic pedagogies and learning theories. We hope to see rich work in these areas with the future view where computing educators would have more explicit arguments why they select some pedagogical choices for their contexts. This might sometimes cause questioning their current implicit assumptions, but if not, making assumptions visible and explicit allows better opportunities to understand what works and why in pedagogy.

Finally, we concur with Malmi et al. [57] and recommend CER publication venues to encourage authors to discuss pedagogical implications as a regular part of discussion, where this is relevant. This would help practicing educators to learn from the research and thus disseminate relevant findings for wider audience.

Appendix A Computing education pedagogical models or techniques identified in the study; ordered alphabetically. Asterisks denote that we found no intersection between the pedagogy and some theory in Appendix B.

Id	Pedagogical model or technique	Explanation	Paper
P1*	Adversarial mindset in cybersecurity	A cybersecurity course design that balances theoretical and practical learning with emphasis on exploring offensive tactics, techniques, and procedures	[65]
P2	Bootcamp	A intensive, specialized, short term training course designed to rapidly prepare students for entering the software industry	[91]
Р3	Computer science unplugged	An approach to teaching computer science concepts through games and puzzles rather than on a computer	[9]
P4	Computing for social-good	A teaching approach using educational activities that emphasize computing for the social good	[33]
P5*	Cryptographic playground	An online environment where students can experiment with and learn about cryptosystems	[49]
P6*	Executable exams	An exam conducted on computer in a programming environment where student can write, compile, run and test their programs	[10]
P7	Hackathon	A fast-paced event where people work collaboratively in teams to create a software application over short period of time	[63]
P8 P9	In-flow peer review (IFPR) Live coding	A peer-review conducted during the development of an assignment A teaching technique where code is written and tested live on a computer in front of students during a class	[13] [78]
P10	Media-based computation	An approach to teaching introductory computing with a media-focused context	[35]
P11	Notional machine	A teaching approach that uses the explicit concept of a notional machine to explain programming constructs or semantics	[23]
P12	Pair programming	A teaching technique where two students work side-by-side at one computer, continuously collaborating on the development of the same program	[8]
P13*	Pair-separate, pair-together, and partner puzzles	A categorisation of three different collaboration modes of novice programmers in a block-based environment	[52]
P14	Parsons problems	An instructional tool for introductory programming where students piece together programming solutions from code fragments	[67]
P15	Peer code review	A collaborative learning approach where students peer review programs written by other students and give feedback	[93]
P16	Plans and goals	An approach to teaching programming that incorporates explicit use of plans and goals	[18]
P17*	Pre-programming Analysis Guided Programming (PAGP)	A structured process to guide students in mimicing how an expert approaches a programming task	[42]
P18	Program tracing	A technique where the programmer traces the execution of program code in order to track the values of variables during execution and to determine the output of the code	[50]
P19	Scrumage (SCRUM for AGile Education)	An agile teaching approach that aims to mimic workplace expectation and promote student autonomy	[28]
P20*	Socially-responsible computing; green education	A teaching model that exposes students to the social impact and ethics of computing	[15]
P21*	Structured mentorship	A mentorship program that combines industry expertise, peer mentorship, and relevant skills, to complement classroom learning	[61]
P22	Tangible; kinesthetic; physical computing	An approach to teaching programming where students arrange and connect physical objects to represent various programming elements, forming physical constructions that describe computer programs	[39]
P23	Use-modify-create (UMC)	A scaffolding framework describing three phases of a learning progression where students use other's code, modify code and then create code Manuscript submitted to AC	[45] M

Appendix B Computing education domain-specific theories and models identified in the study; ordered alphabetically. Asterisks denote that we found no intersection between the theory and some pedagogy in Appendix A.

Theory / Model name	Explanation	Paper
*2DET engagement taxonomy	Extending the engagement taxonomy for program/algorithm visualization to incorporate content ownership dimension	[85]
Abstraction transition taxonomy, ATT	Classification scheme for the knowledge and practices required to apprentice students into the programming community	[17]
Cognitive complexity of computer programs (CCCP)	Framework characterizing the complexity of a program from a cognitive perspective in terms of the hierarchical structure of plans present and their interactions	[27]
*Didactic Focus-based Categorization Method (DFCM)	Scheme for classifying literature based on didactic focuses of educational research	[44]
Engagement taxonomy	Taxonomy of students' interaction with algorithm visualization	[64]
Error quotient (EQ)	Measure of how much a student struggles with syntax errors while programming	[41]
Learner-Directed Model	Pedagogical model combining Kolb's experiental learning and self-regulated learning	[46]
Learning edge momentum	Theory to explain learning progression in introductory programming	[70]
Normalized program state model (NPSM)	Characterization of students' programming activities in terms of the dynamically changing syntactic and semantic correctness of their programs	[11]
Notional machine	Idealized model of the computer implied by the constructs of the programming language	[26]
Predict student success (PreSS#)	Machine learning model that can predict student success early in an introductory programming module (extension of PreSS)	[69]
Programming "geek" gene	Hypothesis of an innate talent for programming	[3]
Programming plans	Expert knowledge as program fragments that represent stereotypic action sequences in programming	[84]
Progression of early computational thinking (PECT) model	Framework for understanding and assessing computational thinking in primary school	[77]
Reducing abstraction	Mental mechanism of reducing abstraction helps students to cope successfully with problems presented to them	[38]
*Simon's scheme	Scheme for classifying CER literature based on Context, Theme, Scope and Nature	[82]
Theory of instruction for introductory programming skills	Holistic theory for teaching programming based on identification of four distinct skills that novices learn incrementally	[96]
Threshold skills	Extension of threshold concepts to include threshold skills	[75]
*Twelve Emotions in Academia Model	Main emotions detected in educational contexts	[74]
*Weighted learning gain (WLG)	Measurement of student learning gains for isomorphic questions in the context of peer instruction	[68]
Zone of proximal flow	Pedagogical design framework integrating Vygotsky's zone of proximal development theory with Csikszentmihalyi's ideas about flow	[6]

Table 4. Appendix C. Frequencies of the intersections between theories and pedagogies. Note that theories and pedagogies that had no intersections have not been included. The 'X' is used to indicate the intersection of notional machine as a theory with notional machine as a pedagogy.

					P	Pedagogies	gies										
Theory / model name	P2	P3	P4	P 7	P8	P9	P10	P11	P12	P14	P15	P16	P18	P19	P22	P23	Total
Abstraction transition taxonomy (ATT)	1	2	١.	-	١.	2	2	9	6	5	1	1	1	1	1	2	29
Cognitive complexity of computer																	
programs	1	ı	•	ı	ı	ı	_	_	1	•	1	1	1	1	1	1	2
Engagement taxonomy	•	1	1	ı	•		\vdash	2	6	2	П	•	П	1	1	1	17
Error quotient (EQ)	٠	1	-	_	1	2	2	2	3	3	2	1	2	1	2	1	20
Learner-Directed Model	٠	1	1	ı	1	,	•	1	_	1	1	1	1	1	1	1	1
Learning edge momentum	٠	3	1	П	ı	4	9	13	26	8	6	1	4	1	П	ı	9/
Normalized program state model (NPSM)	•	1	1	ı	•		•	•	•	1	•	•	1	1	1	1	1
Notional machine	2	10	1	3	ı	10	6	×	14	13	2	1	22	1	7	П	97
Predicting student success	•	١	1	ı	ı	,	٠	•	Τ	П	1	•	•	1	•	1	2
Programming "geek" gene	1	1	1	ı	ı	_	2	2	6	П	2	1	1	1	1	1	18
Programming plans	1	١	1	ı	ı	Н	•	7	4	9	4	П	3	1	1	ı	26
Progression of early computational																	
thinking (PECT) model	•	Т	1	ı	ı	,	٠	•	1	١	1	•	•	1	_	1	2
Reducing abstraction	1	3	1	•	ı	,	1	1	1	١	١	1	1	1	1	П	5
Theory of instruction for																	
introductory programming skills	П	1	1	ı	П	4	2	11	10	17	3	1	19	1	1	П	70
Threshold skills	•	1	1	ı	ı		П	1	3	1	1	1	2	1	1	1	6
Zone of proximal flow	2	2	1	_	1	2	•	3	∞	4	2	1	П	1	1	2	30
Total	2	26	4	7	1	56	56	48	46	61	53	1	54	1	12	7	405

REFERENCES

- [1] [n. d.]. Computer Science without a computer. https://www.csunplugged.org/en/. Accessed: 2024-03-07.
- [2] [n. d.]. Learning theories. http://learning-theories.com. Accessed: 2024-03-16.
- [3] Alireza Ahadi and Raymond Lister. 2013. Geek genes, prior knowledge, stumbling points and learning edge momentum: parts of the one elephant?. In Proceedings of the ninth annual international ACM conference on International computing education research. 123–128.
- [4] Abdullah Al-Sakkaf, Mazni Omar, and Mazida Ahmad. 2019. Social worked-examples technique to enhance student engagement in program visualization. *Baghdad Science Journal* 16, 2 (2019), 0453.
- [5] Mikko Apiola, Mohammed Saqr, Sonsoles López-Pernas, and Matti Tedre. 2022. Computing education research compiled: Keyword trends, building blocks, creators, and dissemination. IEEE Access 10 (2022), 27041–27068.
- [6] Ashok R Basawapatna, Alexander Repenning, Kyu Han Koh, and Hilarie Nickerson. 2013. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In Ninth International Computing Education Research Conference (ICER 2013). ACM, 67–74.
- [7] Ali Battal, Gülgün Afacan Adanır, and Yasemin Gülbahar. 2021. Computer science unplugged: A systematic literature review. Journal of Educational Technology Systems 50, 1 (2021), 24–47.
- [8] Kent Beck. 2000. Extreme programming explained: embrace change. addison-wesley professional.
- [9] Tim Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. Computer science unplugged: School students doing real computing without computers. New Zealand Journal of applied computing and information technology 13, 1 (2009), 20–29.
- [10] Chris Bourke, Yael Erez, and Orit Hazzan. 2023. Executable exams: taxonomy, implementation and prospects. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 381–387.
- [11] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope. 2015. The normalized programming state model: predicting student performance in computing courses based on programming behavior. In 11th International Computing Education Research Conference (ICER 2015). ACM, 141–150. https://doi.org/10.1145/2787622.2787710
- [12] Jacqui Chetty and Duan van der Westhuizen. 2015. Towards a pedagogical design for teaching novice programmers: design-based research as an empirical determinant for success. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. 5–12.
- [13] Dave Clarke, Tony Clear, Kathi Fisler, Matthias Hauswirth, Shriram Krishnamurthi, Joe Gibbs Politz, Ville Tirronen, and Tobias Wrigstad. 2014. In-flow peer review. In Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference. 59–79.
- [14] John Clements and Shriram Krishnamurthi. 2022. Towards a notional machine for runtime stacks and scope: When stacks don't stack up. In Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1. 206–222.
- [15] Lena Cohen, Heila Precel, Harold Triedman, and Kathi Fisler. 2021. A new model for weaving responsible computing into courses across the CS curriculum. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. 858–864.
- [16] Kathryn Cunningham, Barbara J Ericson, Rahul Agrawal Bejarano, and Mark Guzdial. 2021. Avoiding the Turing tarpit: Learning conversational programming by starting from code's purpose. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. 1–15.
- [17] Quintin Cutts, Sarah Esper, Marlena Fecho, Stephen R Foster, and Beth Simon. 2012. The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition. In [Eighth] International Computing Education Research Conference (ICER 2012). ACM, 63-70. https://doi.org/10.1145/2361276.2361290
- [18] Michael De Raadt, Richard Watson, and Mark Toleman. 2009. Teaching and assessing programming strategies explicitly. In Proceedings of the 11th Australasian Computing Education Conference (ACE 2009), Vol. 95. 45–54.
- [19] Peter J. Denning, Douglas E Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R Young. 1989. Computing as a discipline. Computer 22, 2 (1989), 63–70.
- [20] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. 2024. Prompt Problems: A new programming exercise for the generative AI era. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. 296–302.
- [21] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In Proceedings of the fourth international workshop on computing education research. 113–124.
- [22] John Dewey. 1974. John Dewey on education: Selected writings. (1974).
- [23] Paul E Dickson, Neil CC Brown, and Brett A Becker. 2020. Engage against the machine: Rise of the notional machines as effective pedagogical devices. In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. 159–165.
- [24] Stefania Druga, Thomas Ball, and Amy Ko. 2022. How families design and program games: a qualitative analysis of a 4-week online in-home study. In Interaction Design and Children. 237–252.
- [25] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A review of research on parsons problems. In Proceedings of the twenty-second australasian computing education conference. 195–202.
- [26] Benedict Du Boulay, Tim O'Shea, and John Monk. 1981. The black box inside the glass box: presenting computing concepts to novices. *International Journal of man-machine studies* 14, 3 (1981), 237–249.
- [27] Rodrigo Duran, Juha Sorva, and Sofia Leite. 2018. Towards an analysis of program complexity from a cognitive perspective. In Proceedings of the 2018 acm conference on international computing education research. 21–30.

- [28] Shannon Duvall, Scott Spurlock, Dugald Ralph Hutchings, and Robert C Duvall. 2021. Improving content learning and student perceptions in CS1 with scrumage. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. 474–480.
- [29] Barbara J Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S Miller, Briana B Morrison, Janice L Pearce, and Susan H Rodger. 2022. Parsons problems and beyond: Systematic literature review and empirical study designs. Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education (2022), 191–234.
- [30] Katrina Falkner and Judy Sheard. 2019. Pedagogic Approaches. The Cambridge handbook of computing education research (2019), 445-480.
- [31] Peter F Fensham. 2004. Defining an identity the evolution of science education as a field of research. Springer.
- [32] Max Fowler, David H Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West, and Craig Zilles. 2022. Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. Computer Science Education 32, 3 (2022), 355–383.
- [33] Mikey Goldweber, Lisa Kaczmarczyk, and Richard Blumenthal. 2019. Computing for the social good in education. ACM Inroads 10, 4 (2019), 24-29.
- [34] Jean M Griffin. 2016. Learning by taking apart: deconstructing code by reading, tracing, and debugging. In Proceedings of the 17th Annual conference on information technology education. 148–153.
- [35] Mark Guzdial. 2013. Exploring hypotheses about media computation. In Ninth International Computing Education Research Conference (ICER 2013). ACM, 19–26. https://doi.org/10.1145/2493394.2493397
- [36] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. 2011. Pair programming in education: A literature review. Computer Science Education 21, 2 (2011), 135–173.
- [37] Anja Hawlitschek, Sarah Berndt, and Sandra Schulz. 2023. Empirical research on pair programming in higher education: a literature review. Computer science education 33. 3 (2023), 400–428.
- [38] Orit Hazzan. 1999. Reducing abstraction level when learning abstract algebra concepts. Educational Studies in Mathematics 40 (1999), 71–90.
- [39] Michael Horn, Marina Bers, et al. 2019. Tangible computing. The Cambridge handbook of computing education research 1 (2019), 663-678.
- [40] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, et al. 2019. Fostering program comprehension in novice programmers-learning activities and learning trajectories. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education. 27–52.
- [41] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In Second International Computing Education Research Workshop (ICER 2006). ACM, 73–84.
- [42] Wei Jin. 2008. Pre-programming analysis tutors help students learn basic programming concepts. In Proceedings of the 39th SIGCSE technical symposium on Computer science education. 276–280.
- [43] Alark Joshi and Amit Jain. 2018. Reflecting on the impact of a course on inclusive strategies for teaching computer science. In 2018 IEEE Frontiers in Education Conference (FIE). IEEE, 1–9.
- [44] Päivi Kinnunen, Viejo Meisalo, and Lauri Malmi. 2010. Have we missed something? Identifying missing types of research in computing education. In Sixth International Computing Education Research Workshop (ICER 2010). ACM, 13–22. https://doi.org/10.1145/1839594.1839598
- [45] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. Acm Inroads 2, 1 (2011), 32–37.
- [46] Stella Lee, Trevor Barker, and Vivekanandan Suresh Kumar. 2016. Effectiveness of a learner-directed model for e-learning. Journal of Educational Technology & Society 19, 3 (2016), 221–233.
- [47] Alex Lishinski, Jon Good, Phil Sands, and Aman Yadav. 2016. Methodological rigor and theoretical foundations of CS education research. In 12th International Computing Education Research Conference (ICER 2016). ACM, 161–169. https://doi.org/10.1145/2960310.2960328
- [48] Alex Lishinski and Aman Yadav. 2019. Motivation, Attitudes, and Dispositions. The Cambridge handbook of computing education research (2019), 801–826.
- [49] Michael Lodi, Marco Sbaraglia, and Simone Martini. 2022. Cryptography in Grade 10: Core Ideas with Snap! and Unplugged. In Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1. 456–462.
- [50] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the fourth international workshop on computing education research. 101–112.
- [51] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, et al. 2018. Introductory programming: a systematic literature review. In Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education. 55–106.
- [52] Nicholas Lytle, Veronica Cateté, Danielle Boulden, Yihuan Dong, Jennifer Houchins, Alexandra Milliken, Amy Isvik, Dolly Bounajim, Eric Wiebe, and Tiffany Barnes. 2019. Use, modify, create: Comparing computational thinking lesson progressions for stem classes. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education. 395–401.
- [53] Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair. 2019. Computing education theories: what are they and how are they used? In 15th International Computing Education Research Conference (ICER 2019). 187–197.
- [54] Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair. 2020. Theories and models of emotions, attitudes, and self-efficacy in the context of programming education. In 16th International Computing Education Research Conference. 36–47.
- [55] Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair. 2022. Development and use of domain-specific learning theories, models, and instruments in computing education. *Transactions on Computing Education (TOCE)* 23, 1, Article 6 (dec 2022), 48 pages. https://doi.org/10.1145/3530221

- [56] Lauri Malmi, Judy Sheard, Simon, Roman Bednarik, Juha Helminen, Päivi Kinnunen, Ari Korhonen, Niko Myller, Juha Sorva, and Ahmad Taherkhani.
 2014. Theoretical underpinnings of computing education research: what is the evidence? In Tenth International Computing Education Research Conference (ICER 2014). ACM, 27–34. https://doi.org/10.1145/2632320.2632358
- [57] Lauri Malmi, Judy Sheard, Jane Sinclair, Päivi Kinnunen, and Simon. 2023. Domain-Specific Theories of Teaching Computing: Do they Inform Practice?. In Proceedings of the 23rd Koli Calling International Conference on Computing Education Research. 1–15.
- [58] Lauri Malmi, Ian Utting, and Andrew J Ko. 2019. Tools and environments. In The cambridge handbook of computing education research. Cambridge University Press, 639–662.
- [59] Yana Malysheva and Caitlin Kelleher. 2020. Using Bugs in Student Code to Predict Need for Help. In 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 1–6.
- [60] Lauren E Margulieux, Brian Dorn, and Kristin A Searle. 2019. Learning sciences for computing education. Cambridge University Press Cambridge.
- [61] Chao Mbogo. 2019. A Structured Mentorship Model for Computer Science University Students in Kenya. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education. 1109–1115.
- [62] Bhagya Munasinghe, Tim Bell, and Anthony Robins. 2023. Unplugged activities as a catalyst when teaching introductory programming. Journal of Pedagogical Research 7, 2 (2023), 56–71.
- [63] Arnab Nandi and Meris Mandernach. 2016. Hackathons as an informal learning platform. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education. 346–351.
- [64] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J Ángel Velázquez-Iturbide. 2002. Exploring the role of visualization and engagement in computer science education. In ITiCSE 2002 Working Group Reports (ITiCSE-WGR 2002). ACM, 131–152. https://doi.org/10.1145/782941.782998
- [65] TJ OConnor. 2022. Helo darkside: Breaking free from katas and embracing the adversarial mindset in cybersecurity education. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1. 710–716.
- [66] Claudia Ott, Anthony Robins, and Kerry Shephard. 2016. Translating principles of effective feedback for students into the CS1 context. ACM Transactions on Computing Education (TOCE) 16, 1 (2016), 1–27.
- [67] Dale Parsons and Patricia Haden. 2006. 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. 157–163.
- [68] Leo Porter, Cynthia Bailey Lee, Beth Simon, and Daniel Zingaro. 2011. Peer instruction: do students really learn from peer discussion in computing?.
 In Seventh International Computing Education Research Workshop (ICER 2011). ACM, 45–52. https://doi.org/10.1145/2016911.2016923
- [69] Keith Quille and Susan Bergin. 2019. CS1: how will they do? How can we help? A decade of research and practice. Computer Science Education 29, 2-3 (2019), 254–282.
- [70] Anthony Robins. 2010. Learning edge momentum: a new account of outcomes in CS1. Computer Science Education 20, 1 (2010), 37-71.
- [71] Anthony V Robins, Lauren E Margulieux, and Briana B Morrison. 2019. Cognitive sciences for computing education. The Cambridge handbook of computing education research (2019), 231–275.
- [72] David Norman Rodowick. 1997. Gilles Deleuze's time machine. Duke University Press.
- [73] Ma Mercedes T Rodrigo, Thor Collin S Andallaza, Francisco Enrique Vicente G Castro, Marc Lester V Armenta, Thomas T Dy, and Matthew C Jadud. 2013. An analysis of java programming behaviors, affect, perceptions, and syntax errors among low-achieving, average, and high-achieving novice programmers. Journal of Educational Computing Research 49, 3 (2013), 293–325.
- [74] Samara Ruiz, Sven Charleer, Maite Urretavizcaya, Joris Klerkx, Isabel Fernández-Castro, and Erik Duval. 2016. Supporting learning by considering emotions: tracking and visualization a case study. In Proceedings of the sixth international conference on learning analytics & knowledge. 254–263.
- [75] Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, Jan Erik Moström, Lynda Thomas, and Carol Zander. 2012. Threshold concepts and threshold skills in computing. In [Eighth] International Computing Education Research Conference (ICER 2012). ACM, 23–30. https://doi.org/10. 1145/2361276.2361283
- [76] Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, and Carol Zander. 2017. Folk pedagogy: Nobody doesn't like active learning. In Proceedings of the 2017 ACM Conference on International Computing Education Research. 145–154.
- [77] Linda Seiter and Brendan Foreman. 2013. Modeling the learning progressions of computational thinking of primary grade students. In Ninth International Computing Education Research Conference (ICER 2013). ACM, 59–66. https://doi.org/10.1145/2493394.2493403
- [78] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live coding: A review of the literature. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. 164–170.
- [79] Sue Sentance, Jane Waite, and Maria Kallia. 2019. Teaching computer programming with PRIMM: a sociocultural perspective. Computer Science Education 29, 2-3 (2019), 136–176.
- [80] Lee Shulman. 1987. Knowledge and teaching: Foundations of the new reform. Harvard educational review 57, 1 (1987), 1–23.
- [81] Lee S Shulman. 2005. Signature pedagogies in the professions. Daedalus 134, 3 (2005), 52-59.
- [82] Simon. 2007. A classification of recent Australasian computing education publications. Computer Science Education 17, 3 (2007), 155–169. https://doi.org/10.1080/08993400701538021
- [83] Simon. 2015. Emergence of computing education as a research discipline. Ph.D. Dissertation. Aalto University.
- [84] Elliot Soloway and Kate Ehrlich. 1984. Empirical studies of programming knowledge. IEEE Transactions on software engineering 5 (1984), 595-609.

- [85] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. ACM Transactions on Computing Education (TOCE) 13, 4, Article 15 (Nov. 2013), 64 pages. https://doi.org/10.1145/2490822
- [86] Claudia Szabo, Nickolas Falkner, Andrew Petersen, Heather Bort, Kathryn Cunningham, Peter Donaldson, Arto Hellas, James Robinson, and Judy Sheard. 2019. Review and use of learning theories within computer science education research: primer for researchers and practitioners. In ITiCSE 2019 Working Group Reports. 89–109.
- [87] Claudia Szabo and Judy Sheard. 2023. Learning theories use and relationships in computing education research. Transactions on Computing Education (TOCE) 23, 1, Article 5 (Mar 2023), 34 pages. https://doi.org/10.1145/3487056
- [88] Matti Tedre and John Pajunen. 2022. Grand theories or design guidelines? Perspectives on the role of theory in computing education research. ACM Transactions on Computing Education 23, 1 (2022), 1–20.
- [89] Matti Tedre, Simon, and Lauri Malmi. 2018. Changing aims of computing education: A historical survey. Computer Science Education 28, 2 (2018), 158–186.
- [90] Matti Tedre and Erkki Sutinen. 2008. Three traditions of computing: what educators should know. Computer Science Education 18, 3 (2008), 153-170.
- [91] Kyle Thayer and Amy J Ko. 2017. Barriers faced by coding bootcamp students. In Proceedings of the 2017 ACM Conference on International Computing Education Research. 245–253.
- [92] Kristina von Hausswolff. 2022. Practical thinking while learning to program–novices' experiences and hands-on encounters. Computer Science Education 32, 1 (2022), 128–152.
- [93] Yanqing Wang, Hang Li, Yuqiang Feng, Yu Jiang, and Ying Liu. 2012. Assessment of programming language learning based on peer code review model: Implementation and experience report. Computers & Education 59, 2 (2012), 412–422.
- [94] Meredith Williams. 1999. Wittgenstein, mind and meaning. Toward a social conception of mind (1999).
- [95] Krissi Wood, Dale Parsons, Joy Gasson, and Patricia Haden. 2013. It's never too early: pair programming in CS1. In Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136. 13–21.
- [96] Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. 2019. A theory of instruction for introductory programming skills. Computer Science Education 29, 2-3 (2019), 205–253.