15 Pedagogic Approaches

Katrina Falkner and Judy Sheard

15.1 Why Is Pedagogy Important?

Over the last couple of decades, there has been a shift in focus in our education systems from teacher-centered to student-centered education and increasing use of technology. These changes have provided impetus for the development and adoption of new and different pedagogic approaches. There are many factors that influence the choice of pedagogy, including contextual and discipline-based factors. These are important to understand as the pedagogic approach has direct and profound implications for student learning and the student learning experience.

Within the computing discipline, pedagogic approaches are of great concern. There are specific learning challenges in computing associated with comprehension of complex concepts, analytical and problem-solving skills, and implementation of a solution in a programming language. As evidence of these challenges is the extensive literature on students' difficulties with learning how to program. As a result, computing programs are often reported, although somewhat contentiously, as having high attrition and high failure rates (Watson & Li, 2014). Another concern related to pedagogic approaches is the lack of diversity within our student cohort, and in particular low numbers of female students.

The rapid evolution of computing poses additional challenges. To address industry needs, student preference, and pedagogic improvement, educators often choose to incorporate constant change in the programming languages, paradigms, platforms, and technologies they use. A recent challenge is the sharp increase in enrollments in many computing courses; at the same time, however, there are signs of decreasing student engagement. Although there has been considerable effort invested into adapting to students' learning needs, there is an apparent mismatch between the pedagogic approaches we use to teach our students and how they want to learn.

There are huge opportunities afforded by technology that can be used to address many teaching and learning issues and bring many benefits to students. Within the computing discipline, we have the knowledge to design, build, and adapt technology for teaching and learning. It is essential, however, to understand how technology can best enable or be effectively incorporated into a pedagogic approach, as the use of technology without clear purpose or adequate pedagogic support can introduce further risk of reducing student engagement (Preston et al., 2010).

https://doi.org/10.1017/9781108654555.016

With the range of pedagogic approaches we now have available, it is important that we are able to share practices that address the unique challenges of our discipline, increase accessibility and success in the learning of computing, and ensure an engaging and valuable learning experience for our students.

15.2 Background

Pedagogy is the science of how we promote learning and it consists of the learning activities, strategies, and techniques that provide the environment where learning may take place. Pedagogy is a widely applied but complex concept that includes relationships between the teacher, students, activities, and learning context. An important distinction is that pedagogy is about *how* teaching is done, rather than *what* is taught, while noting that one may influence the adoption of the other.

The term "pedagogy" can be applied to the educational approach used in a range of contexts from an individual activity to an entire course or teaching program. For the purposes of this chapter, we will use *pedagogy* when describing a set of activities and strategies that guide the teaching of a course. A pedagogy may define a specific approach (e.g., *blended learning*, a specific approach blending modes of learning) or a broad range of approaches (e.g., *active learning*, describing a number of approaches built around active student participation). We will use *pedagogic practice* when describing a specific activity within a course (e.g., *peer instruction*) (Simon & Cutts, 2012). In some cases, pedagogy will be used as an umbrella term for a set of pedagogic practices (e.g., contributing student pedagogy) (Collis & Moonen, 2005; Hamer et al., 2008).

There are many influences on the development of a pedagogy and how it is applied (Firmin et al., 2012). Some of these influences relate specifically to the discipline in which they are used, and this has led to the concept of *signature* pedagogies (Shulman, 2005). In our discussion, we will focus on pedagogies that are used in the computing discipline, and some of these will be specific to the computing discipline.

The two prominent theoretical positions that have influenced conceptions of learning are derived from *behavioral* and *cognitive* psychology. These provide quite different perspectives from which to understand the nature of learning. Behavioral psychologists view learning as directly observable behavior that can be measured by behavioral responses in the learner. Learning is seen as a process of conditioning by instruction that can guide and shape the learning through sequences of *stimuli*, *responses*, *feedback*, and *reinforcement* (Phillips & Soltis, 2015). Behaviorists equate learning with observable behavioral outcomes and do not consider the role of mental operations in the learning process. In contrast, cognitive psychologists focus on the mind as the agent of learning. They are concerned with the internal mental constructions and activities of the learner in preference to their external behavior. Cognitive psychologists stress the importance of learning through a variety of learning strategies that depend on the type of learning outcomes desired, reflecting associated cognitive

processes. These strategies include *memorization*, *drill and practice*, *deduction*, and *induction* (Atkins, 1993; Jonassen, 1991; Reeves & Reeves, 1997).

Behaviorism and cognitive psychology are often presented as opposing philosophies in their explanations of learning. However, there are similarities in that both seek to effect learning through the design of specific tasks. The main differences are the foci on either external or internal activities of the learner and the epistemologies on which they are based (Jonassen, 1991). The behaviorist position draws on an *objectivist* view of knowledge. The central tenet of *objectivism* is that there exists a reality with a structure that can be assimilated by the learner. The role of the educator is to interpret, model, and present this reality to their students (Jonassen, 1991). From behaviorism and objectivism the *instructivist* approach to teaching and learning has evolved. This approach focuses on the structure and presentation of the learning material rather than the learners who act as recipients of the instruction.

In contrast, cognitive psychology focuses on the cognitive activity of the learner. Cognitive psychology encompasses a broad range of views on the structure of knowledge and the processes involved in learning. These range in a continuum from a view that learning is affected and monitored by external stimuli and mediated by cognitive processes to a view that learning is a cognitive process initiated by the learner. The first view is linked to an objectivist epistemology and the second to a *constructivist* epistemology (Lowyck & Elen, 1993).

Constructivism states that "students construct knowledge rather than merely receive and store knowledge transmitted by the teacher" (Ben-Ari, 1998). From a constructivist perspective, students learn by relating new concepts to existing ones, either by combining new knowledge with existing knowledge to create new cognitive structures or through reflection upon existing knowledge. Duffy and Cunningham (1996) propose that two commonly agreed-upon tenets of constructivism are as follows (p. 171):

- Learning is an active process of constructing rather than acquiring knowledge.
- Instruction is a process of supporting that construction rather than communicating knowledge.

Constructivist pedagogy encourages students to engage with and develop ownership over their learning processes, encouraging reflective and deep learning through supporting critical thinking, transferability, and self-directedness. In its simplest form, a constructivist pedagogy integrates student activities into the classroom (e.g., by incorporating group discussions, quizzes, or silent reflection designed to encourage engagement with learning objectives).

There has been significant research into the application of constructivist learning strategies, in both small and large classes, with the majority of studies identifying significant benefit in learning outcomes (Ben-Ari, 1998; Geer & Rudge, 2002; Prince, 2004). Prince (2004) undertakes a review of constructivist learning studies within Engineering or related courses, identifying that considerable evidence exists for the core elements of these approaches, while noting that results across the numerous studies vary in strength and the difficulty in assessing and measuring complex pedagogic approaches.

Constructivism is often classified into two views providing related but in some respects complementary perspectives (Duffy & Cunningham, 1996). One view is *cognitive constructivism*, an individual view of constructivism where knowledge is not transferred from one person to another, but in contrast, individuals construct knowledge by making connections between new experiences and established ideas (based on the work of Piaget, 1972, and Papert, 1993), and the other view is *sociocultural constructivism*, where knowledge is created through social and cultural activation (Ernest, 1995). Theories of sociocultural constructivism and community-based learning build upon the ideas that students engage more deeply with their learning processes when actively involved and when learning as part of a group.

Vygotsky (1978) elaborates the ideas of sociocultural constructivism through his observations of the *zone of proximal development*: that learning awakens a variety of internal development processes that operate only when one is working with others; that is, learning is achieved within a social context (a pairing or group) and as a direct result of the group activities. He viewed learning as a profoundly social process and emphasized the importance of the learner–instructor and learner–learner dialogue in order to help the learner progress through their zone of proximal development. He maintained that this is more than just exposing the learner to new material through the provision of resources or lectures; it is through dialogue with others that the learner constructs their understanding. The essential point is that learning is a mediated process. Learners are able to challenge their thoughts, beliefs, perceptions, and existing knowledge by collaborating with other students (Ewing, Dowling, & Coutts, 1998). In this model, learning of new concepts and processes is achieved by first establishing mastery as a group, and only then developing mastery as an individual through continued practice and internalization of knowledge.

Wenger (1998) also describes the importance of learning communities through his definition of communities of practice: communities of peers gathering to share and develop knowledge in a common context. In these communities, the roles of teacher and student are not fixed, but move around the group according to the current focus and expertise provided by individuals. This model of community-based learning echoes practice within industry, where cooperation with colleagues is suggested as a key professional skill and is also a key requirement in achieving expertise in software development (Sonnentag, 1998).

Modern pedagogic practice has tended toward constructivist or student-centered approaches over instructivist approaches. Approaches that support constructivist learning are frequently known as *active* learning approaches. Active learning pedagogic practices designed to support *cognitive constructivism* range from small-scale active learning practices that may be integrated into broader learning opportunities, such as minute papers and mind mapping (Phillips & Soltis, 2015), through to problem-based or enquiry-based learning approaches, where learning is driven by student motivation via the problem-solving process.

Sociocultural constructivist perspectives can be supported by a range of active, social pedagogic approaches, including collaborative learning, cooperative

learning, contributing student pedagogy, blended learning, and massive open online course (MOOC)-based pedagogic approaches.

15.2.1 The Influence of Technology on Pedagogy

The increased integration of technology within our learning environments has inspired the development of pedagogic practices that take advantage of the affordances of technology. These technology-enabled and technology-enhanced pedagogic practices have had a profound impact on the teaching and learning of computing.

Advances in information, network, and communication technologies have enabled the development of online educational platforms and led to the proliferation of online learning resources. Increasingly, learning material and activities used in face-to-face mode are being made available online. In addition, technology has afforded new forms of resources, leading to innovative pedagogic practices. For example, the use of social media in education has led to new forms of learning activities and assessment (e.g., Vickers et al., 2014; Vozniuk, Holzer, & Gillet, 2014).

The ready availability of computing and communication devices means that students are now able to access learning resources and communicate with teaching staff and other students while off campus and at any time. This has inspired pedagogic approaches that recognize that learning now happens in many situations. Arguably the most widely used example is *blended learning*, where a course is designed with both face-to-face and online learning activities, with the educator selecting the most appropriate mode for each activity to achieve the learning outcomes. A pedagogic practice that draws on this approach is the *flipped classroom* (Bishop & Verleger, 2013).

Other technology innovations have inspired or enhanced pedagogic practices. A prominent example is MOOCs, which have motivated new pedagogic practices associated with blended and online learning. Another example is the use of learning analytics, which have led to pedagogic practices that enable students to monitor and understand their learning progress.

15.3 Implications for Practice within Computing

There exists a wide range of pedagogic approaches within the computing discipline supported, in turn, by a number of associated pedagogic practices. For example, in their recent survey, Sanders et al. (2017) identify 38 different forms of active learning in the computing literature alone, some building on well-established techniques from other disciplines, while others are specific to computing, often building from industry practice. Table 15.1 provides a representation of the key pedagogic approaches we are exploring in this chapter, along with relevant examples of pedagogic practice from the computing literature.

Implementation of pedagogic practice typically occurs within a real-world context, with matters such as physical space availability, technical capabilities,

Table 15.1 *Pedagogic approaches and relevant pedagogic practices.*

Pedagogy	Pedagogic practices	Description
Active learning	E.g., Parson's problems (Parsons & Haden, 2006); in-lecture activities or e-tivities (Salmon, 2004); test-driven development (Beck, 2002); test-first development (Politz, Krishnamurthi, & Fisler, 2014); live coding (Rubin, 2013)	General term to describe a range of practices where students are involved in actively *doing* and *reflecting* to facilitate their learning
Collaborative learning	E.g., peer instruction (Simon & Cutts, 2012); studio-based learning (Hundhausen, Narayanan, & Crosby, 2008)	General term to describe a range of practices where students collaborate in the learning process
Cooperative learning	E.g., Jigsaw (Aronson et al., 1978); pair programming (Williams & Kessler, 2002)	General term to describe a range of practices where students collaborate and have accountability for group learning
Contributing student pedagogy	E.g., content creation; activity creation; peer assessment or review (Earl, 1986)	General term to describe a range of collaborative practices where students produce valued artifacts for the purpose of contributing to other students' learning
Blended learning	E.g., Flipped classroom (Bishop & Verleger, 2013)	General term to describe a range of active instructional practices that blend modes of learning, typically online and face-to-face. Can be either individual or collaborative, but primarily collaborative
MOOC	E.g., xMOOC (Glance, Forsey, & Riley 2013); cMOOC (Siemens, 2005); hybrid models	General term to describe pedagogic approaches built on top of the MOOC format; used in other pedagogic practices

and budgetary or staffing considerations often constraining or influencing adoption and adaptation. It is common for implementations as described in the literature to express variations incurred from these or similar factors, and for characteristics from multiple pedagogic practices to be adopted together in order to meet to needs of the specific environmental context or cohort. An example of this is the work by Pollock and Harvey (2011), which integrates

studio-based learning (Hundhausen, Narayanan, & Crosby, 2008), problem-based learning (Barrows, 1986), and active learning practices as a single practice. Another example is a CS1 course where a trio of pedagogic practices – media computation (Rich, Perry, & Guzdial, 2004), pair programming (Williams & Kessler, 2002), and peer instruction (Simon & Cutts, 2012) – were used in order to improve the quality of the course, improve retention, and appeal to a broader cohort of students (Porter & Simon, 2013)

Within our discussion here, we will attempt to provide a general overview of each pedagogic approach and include examples specific to computing, or applied within computing, of relevant pedagogic practices.

15.3.1 Active Learning (Cognitive Constructivist Approaches)

Sanders et al. (2017) identify that "active learning" is a poorly understood but widely used pedagogic term within the computing discipline, encompassing a wide range of pedagogic approaches with variation in their underpinning philosophy. They define active learning as having the following two key elements: students actively undertaking an activity and having an opportunity to think and/or reflect about their learning as part of the process. However, while most examples of active learning are indeed "active," not all encompass opportunities for reflection.

Prince (2004) provides a cross-discipline overview of active learning techniques and their effectiveness, while Hativa (2000) presents a general discussion of active learning activities that are suitable for integration within lectures (both cognitive and sociocultural constructivist).

Active learning practices are often, but not always, social. We draw a distinction between those active learning techniques that are based on a cognitive constructivist approach versus those that integrate elements of social or collaborative learning. Cognitive constructivist approaches to active learning provide opportunities for students to individually apply, test, and reflect on their learning and may vary in context, duration, and complexity.

Kurtz et al. (2014) present an example active learning pedagogic practice that requires students to take part in brief, tablet-based active learning activities during lectures. In this example, students undertake a mixture of *logical microlabs*, which require them to solve an abstract problem, submitted graphically via their tablet, and *code magnet microlabs*, where students are asked to solve the same problem, but this time programmatically. The approach used by Kurtz et al. creates a learning cycle where, after a period of lecture exploring new concepts, students firstly complete a logical microlab to facilitate understanding of the problem space, followed by a worked example *invention* phase where the instructor models the development of algorithmic understanding from the abstract problem, through to concrete implementation in the code magnet microlab. Students are given individual, immediate, high-level feedback via the assessment tools on their tablet.

Ebert and Ring (2016) present a similar pedagogic practice based on structured live-coding activities within lectures, building on a web-based

presentation framework that allows sharing of developed software solutions within the lecture session. A similar, less technologically enhanced approach has been explored by Falkner and Palmer (2009).

Parsons and Haden (2006) introduced the idea of *Parsons' problems*, an active learning approach where students solve programming puzzles in which lines of code that represent a correct, well-formed solution to a programming problem must be placed in the correct order. Parsons' problems promote understanding of programming construction while enabling students to focus on program semantics rather than syntax. Parsons' problems constrain the logic choices within the program solution, providing enhanced scaffolding. The inclusion of *distractors* – incorrect lines of code included in the set of possible code segments – helps focus students on common errors while still providing constraint over the overall solution space. Web-based implementations provide automated and immediate feedback. Sirkiä (2016) combines Parsons' problems with algorithmic visualization, demonstrating potential for increased understanding through exposure of the notional machine (du Boulay, 1986; see also Section 15.4.4 and Chapter 12 for further discussion).

Active learning, both individual and collaborative, within the computing discipline can benefit greatly from inspiration from industry practice, adopting and adapting active application models used within industry to facilitate learning, while also exploring and building capacity in industry norms. Test-first development is an active learning pedagogic practice that applies the writing of software tests to guide the software development process (Politz et al., 2014), while test-driven development is a similarly active learning practice with a more formal process and structure for test and associated code development (Beck, 2002). Edwards (2004) argues for the benefits of test-driven development in building self-reflection skills, while Politz et al. (2014) explore the combination of peer review and test-first development through their construction of *in-flow* peer review of tests, identifying a causal impact on student learning improvement. Paul (2016) explores the explicit pedagogic practice of test-driven development activities as a way of exploring and understanding the problem space prior to undertaking the software development process.

15.3.2 Collaborative Learning

Collaborative learning involves students working together to informally facilitate their learning, embodying the sociocultural constructivist perspective. Collaboration in the computing classroom can take the form of the adoption of general collaborative practices such as incorporating activities like *think–pair–share* or scaffolded discussions in classroom settings (Hativa, 2000), technology-assisted collaborative note-taking (Jonas, 2013; Reilly et al., 2014), and computing-specific collaborative activities, including peer instruction (Simon & Cutts, 2012), studio-based learning (Hundhausen, Narayanan, & Crosby, 2008), and peer reviews (Clarke et al., 2014; Hundhausen, Agrawal, & Agarwal, 2013; Politz, Krishnamurthi, & Fisler, 2014).

Peer instruction involves students collaboratively problem-solving, with the aim being for students to engage in the practice of explaining core concepts as they collectively attempt to solve problems. Key aspects of the peer instruction practice include pre-review of content, with a pre-quiz on content knowledge to motivate and prime students with relevant content knowledge, and the development of a suite of problems that require students to demonstrate deep understanding of key concepts in order to approach a correct solution. Peer instruction has been shown to significantly improve student learning outcomes; as a key pedagogic practice within computing, peer instruction is discussed in more detail with a focus on students as teachers in Chapter 29.

Studio-based learning (Hundhausen, Narayanan, & Crosby, 2008) is a collaborative learning approach based on the idea of students working within a shared design space, supported frequently by technology aiding collaboration, discussion and sharing of application experiences. Students are typically presented with a substantial problem or series of related problems and iterate between a number of structured phases. These phases include independent design and development (within the *design studio*) and presentation and collaborative peer review within a design critique phase. Informal collaboration may occur within the independent stages, as needed and instigated by the learners; however, collaboration is critical to the success of the presentation and critique phase.

Studio-based implementations can vary in the degree of scaffolding and structure, however. Hundhausen, Narayanan, and Crosby (2008) identify the following two key elements that are common to all approaches:

- Representation construction, where students define their own visual presentation representations for their work (e.g., code, visualizations or algorithms);
- Representation presentation and discussion, where design critiques are used for presentation and discussion as a collaborative exercise.

Hundhausen, Narayanan, and Crosby (2008) identify that studio-based learning provides benefit through scalability, adaptability, and independence from technology, as students explore their own model of visualization or representation for their work, which may range from the use of art supplies through to sophisticated algorithm visualization technology. In their description, multiple implementation examples are explored, demonstrating the flexibility within the pedagogic practice and the variation in dependence on collaboration.

Carbone and Sheard (2002) describe a studio-based learning approach within a first-year IT context, adopting a studio model where students work across four studio spaces: a design studio dedicated to fundamental skill development, where students work either independently or in small groups; a second design studio dedicated to teamwork and large-scale discussions; a cafe-style informal meeting place facilitating reflection and informal social time; and a meeting room for more structured meetings, presentations, and consultations.

Hundhausen, Agrawal, and Agarwal (2013) extend the studio-based learning practice with the integration of pedagogic code reviews. In this extension, a

code review process based on industry-inspired formal code inspection is part of the design critique phase. This extension expands on the opportunity to build communication, collaboration, feedback, and reflection skills available within the design critique phase with industry-based skills specific to the computing discipline.

15.3.3 Cooperative Learning

Cooperative learning (Slavin, 1991) is a sociocultural constructivist approach where students work in small groups or teams to help each other learn and are also responsible for each other's learning (in contrast to collaborative learning). In cooperative learning pedagogic practices, students are encouraged to discuss and debate discipline concepts, with a focus on assisting all students in the team with their learning. Typically, cooperative learning examples incorporate elements of individual accountability toward achieving group goals (i.e., the success of the team depends on individual learning, meaning that it is in each student's interest to facilitate the learning of others). In some cooperative learning approaches, the assessment itself is aligned with cooperative principles, with the group being assigned recognition or assessment grades based on the average academic performance of each member (Slavin & Cooper, 1999).

An important aspect of many cooperative learning implementations is that they embody *equal opportunities for success* (Slavin, 1991). In this approach, individual success is determined by improvement from past assessment exercises, providing each student with the opportunity to succeed. Slavin and Cooper (1999) have explored this aspect of cooperative learning as a means for managing and enhancing academic diversity in the classroom, as each student, regardless of their experience or capability, can contribute to the success of the group, with a reduction in competitiveness and individualism.

Portillo and Campos (2009) describe an implementation of the *jigsaw* cooperative learning technique applied to the creation of analysis class diagrams. The jigsaw technique (Aronson et al., 1978) consists of students working in teams, where each member of the team is assigned a specific task. Across the class, all students assigned the same task, who subsequently have the most knowledge of the class relevant to that task, meet in a so-called *expert group* to discuss and refine their understanding further. Each student then returns to their original group, where they are responsible for teaching their teammates the relevant aspects of their task. In Portillo and Campos's approach, each member of the team is asked to complete an analysis class diagram for a different use case specification, with expert teams being formed around each use case.

Beck and Chizik (2013) describe the use of cooperative learning within an introductory CS1 course using an approach similar to process-oriented guided-inquiry learning (POGIL) (Moog & Spencer, 2008). In this approach, the instructor takes on the role of leader and assists learning by asking questions that prompt deeper thinking or reflection. In Beck and Chizik's approach, there is an emphasis on evaluation, involving the whole class in questioning and

discussion during the evaluation phase, to increase learning and aid reflection. Students are assigned roles during each activity, which are rotated in subsequent activities to provide a broad learning experience. Different forms of cooperative learning are applied in this approach, with students working in groups at times and in pairs at others.

Pair programming (Williams & Kessler, 2002) involves students working in pairs, with one member *driving* the cooperation by leading the typing or writing, while the other – the *navigator* – observes the driver, asking questions and making suggestions. These roles are switched at structured points. Williams et al. (2008) have identified a set of key guidelines for implementing pair programming within the classroom based on several years of experience in classroom application. Research has demonstrated that pair programmers are more effective, producing higher-quality outcomes in less time (Williams, 2000). A meta-analysis of 18 pair programming studies in an educational setting with a total sample size of 3,308 students found significant and positive effects on students' grades on programming assignments and exams and on student persistence (pass rates), but there was no significant effect on students' attitudes (Umapathy & Ritzhaupt, 2017). As a key pedagogic practice with students as teachers within the computing discipline, pair programming is explored further in Chapter 29.

15.3.4 Contributing Student Pedagogy

Contributing student pedagogy (CSP) (Collis & Moonen, 2005; Hamer et al., 2008) is a sociocultural constructivist approach that places the student firmly at the center of learning, with an emphasis on students using, and explicitly valuing, the products of other students' learning. Examples of pedagogic approaches supporting a CSP perspective include peer assessment and supplemental instruction.

In Hamer et al. (2008), CSP is defined as "A pedagogy that encourages students to contribute to the learning of others and to value the contribution of others," and this definition was subsequently refined as "A pedagogy that requires students to produce an artifact for the purpose of contributing to other students' learning, and encourages students to value these peer contributions" (Hamer et al., 2011). Key features of this refinement are (Hamer et al., 2011) as follows:

- The instructor is explicitly using a contributing student pedagogic design.
- The pedagogy includes the requirement that students produce artifacts.
- The students are aware that the audience for these artifacts includes other students.
- These artifacts are shared with other students for the purposes of their learning.

Peer assessment (Earl, 1986) involves peers reviewing, assessing, and providing feedback on student contributions and motivates students to engage in critical analysis skills. Purchase (2000) describes a peer assessment where students assess

software artifacts (in this case, interface designs) produced by other students. Purchase introduces a training exercise encompassing the requirements and boundaries of peer review, including specific assessment criteria.

Peer review has been extended in several studies to software testing, including Smith et al.'s (2012) study, where a peer testing component is added to programing assignments within an introductory data structures course. Students first produce their own solution, then move to evaluating and testing other students' work, including the production of a testing methodology report, before the final phase of each student reviewing and reflecting on their own experience and the artifacts produced by their peers in the production of a peer testing report evaluating the process as a whole.

Politz, Krishnamurthi, and Fisler (2014) present an example of *in-flow peer review* within an online learning environment that incorporates test review and peer review while the student is developing their solution, rather than afterwards. In their approach, students are required to submit their software implementations and test suites at multiple points within the development process. Initial submissions at each point are then submitted to other students for review, judging both correctness and thoroughness. Students are only able to submit at later points if they have submitted all of their required reviews. With peer review occurring throughout the development process, it is more likely to be considered useful and to be used by peers (Clarke et al., 2014). Students are also able to establish a comparative baseline with others' work and to receive early, detailed feedback on misconceptions or poorly thought-out strategies. A detailed analysis of in-flow peer review and the choices available when implementing this pedagogic practice, including the critical aspect of rubric usage, is provided in Clarke et al. (2014).

Content creation, similar to supplemental instruction (Blanc, DeBuhr, & Martin, 1983), involves students learning through the development of teaching materials. Ching et al. (2005) explore a graduate class where students prepare instructional materials for a tutorial topic, which are then evaluated by their peers. The time required to prepare high-quality materials is recognized by the students involved, as are the benefits resulting from this effort. Nortcliffe (2005) describes an approach where students work collaboratively in designing and developing instructional materials for a web design course. Nortcliffe introduces an assessment function that incorporates individual and group performance and peer and self-assessment.

Further examples of content creation include the collaborative preparation of learning resources, such as glossaries and knowledge bases for discipline topics, frequently supported by online tools such as wikis and blogs (Elgort, Smith, & Toland, 2008; Lutteroth & Luxton-Reilly, 2008), and the development of algorithm visualizations (Crescenzi & Nocentini, 2007).

Activity creation involves students preparing learning activities that other students are expected to undertake, such as the creation of physical sorting games to illustrate the sorting method (Gehringer & Miller, 2009), as well as the learning materials on a core discipline concept, matched with an online

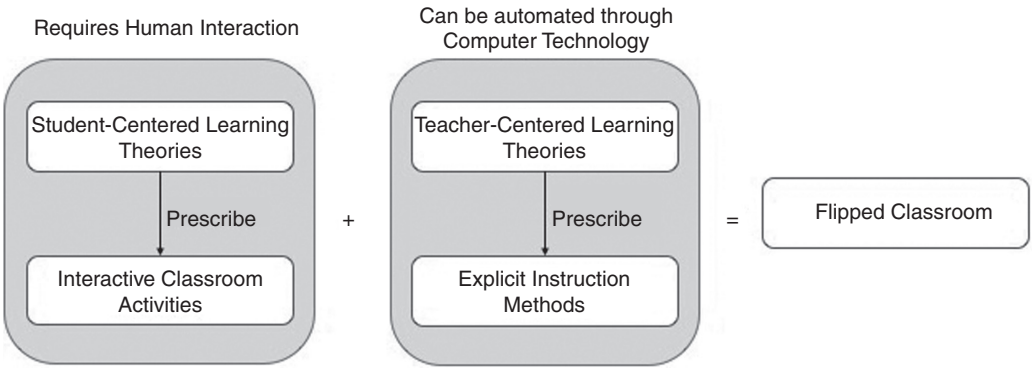

Figure 15.1 *The combination of student-centered learning activity and teacher-centered instruction as adopted within blended learning (adapted from Bishop & Verleger, 2013).*

quiz (Abad, 2008). PeerWise (Denny, Luxton-Reilly, & Hamer, 2008) is an activity creation practice that engages students to create multiple-choice questions as an assessment task, including the definition of multiple alternative answers and an explanation of why the correct answer is correct. Students are able to engage with the system to attempt to answer questions and are given the option of rating the difficulty of the question and providing comments. PeerWise promotes critical analysis, and research has shown that there is a positive correlation between activity within the PeerWise system and learning outcome (Luxton-Reilly et al., 2012). Studysieve (Luxton-Reilly, Plimmer, & Sheehan, 2010) is an extension of this model, supporting questions with free-text answers.

Kaczmarczyk, Boutell, and Last (2007) describe an approach combining content and activity creation within a CS1 course, where students are required to prepare instructional materials in the form of a *capsule* of learning designed to address one of a specific set of topics and including both content and activity design, assessment, and discussion prompts.

15.3.5 Blended Learning

Blended learning, implemented via the flipped classroom pedagogic practice (Bishop & Verleger, 2013), takes advantage of the availability of online or established resources, such as MOOCs, to *flip* the learning model toward an active pedagogic approach. In this model, face-to-face instructional time is focused on active or collaborative learning activities designed to encourage the application of learning, while access to discipline content, provided by existing resources, is undertaken independently, and ideally prior to the opportunities for application.

Figure 15.1 illustrates the relationship between student-centered (constructivist) and teacher-centered (instructivist) approaches within blended learning.

In this perspective, ownership of learning is placed within the student's context, with motivation for learning and engagement with resources being driven by active engagement with the application of knowledge. While flipped classroom pedagogies do not solely rely on the existence of online learning resources, their ease of use when these resources do exist has made this an increasingly prominent approach.

The flipped classroom pedagogy has been used extensively in the computing discipline. In a systematic review of the research on flipped classrooms by Giannakos, Krogstie, and Chrisochoides (2014), almost half of the studies reported (15 out of 32) were from computer science courses. More recent reports provide evidence of widespread uptake in computer science courses. For example Maher et al. (2015) describe strategies for flipped classroom implementations in four different computer science courses.

The terms "flipped" or "inverted" may imply a simple swapping of activities; however, in many cases, the implementation of a flipped classroom results in more complex pedagogic changes, with most involving technology. The review by Giannakos et al. (2014) found a variety of technologies used to flip the classroom. Video lectures were the most frequently used technology, but there were also a number of examples of animated readings and simulations. Online quizzes were used extensively either during or outside classes. Despite the emphasis on technology, a study (n = 284) by Tarimo, Deeb, and Hickey (2015) that compared outcomes from computer-based and pen-and-paper-based interactive classes found that while students preferred in-class engagements and interactions using computers, there were no significant differences in learning outcomes or the students' enjoyment of the material. A smaller study (n = 47) by Lacher and Lewis (2015) on the effectiveness of gate-check video quizzes found no differences in grades between the students who took the quizzes and those who did not.

Many studies have reported positive effects of flipped classrooms. The review by Giannakos et al. (2014) found benefits for students' performance, attitudes, and engagement. An early comparative quasi-experimental study (n = 46) by Day and Foley (2006) of flipped classroom and traditional offerings (control) of an introductory human–computer interaction class found that the students in the flipped classroom gained significantly higher grades and reported stronger positive attitudes about their experiences in contrast to the control group. A larger, more recent comparative study (n = 1901) by Horton et al. (2015) also found that students in the flipped classroom gained significantly higher grades; however, there were no differences in the pass rates and enjoyment of their course between this group and the control group. A smaller quasi-experimental study (n = 65) by Reza and Baig (2015) that investigated the effectiveness of a flipped (inverted) classroom model in a data structures and algorithms course found that students in the flipped classroom had higher levels of programming and problem-solving abilities than students in the traditional classroom, but there was no difference in their theoretical knowledge. Titterton, Lewis, and Clancy (2010), in their study of

three blended learning cohorts, identified that increased learning outcomes were not consistent across all forms of assessment, with no significant difference in self-reported procrastination.

A flipped classroom pedagogy can present challenges to instructors and their students. Giannakos et al. (2014) found the main difficulties faced by instructors were that they spend more time in preparation, the students do not always respond well to the flipped classroom model, and there can be a decrease in class attendance. Students have reported negative experiences and impressions of the flipped classroom pedagogy. A flipped classroom typically requires students to prepare before class and be actively engaged in class, which some can find challenging. Lockwood and Esselstein (2013), reporting experiences in a flipped introductory programming class, found that some students struggled with time management and the requirement to attend class regularly. Maher et al. (2015), describing their experiences flipping four different courses, found from student feedback that some students felt that not enough time was spent on in-class teaching.

15.3.6 MOOC-Based Pedagogy

MOOCs have been adopted as a means to deliver content (usually freely) across distributed environments to anyone with an Internet connection and computer, often supported by social media structures that offer opportunities for collaboration and knowledge sharing, despite learner locations. MOOCs offer one means to deliver education at a broad scale to individuals with technological means and Internet access. Although online learning is not new, it has been argued that the difference between online learning and MOOC environments are the combination of teaching approaches course instructors use, the massive levels of participation, and their openness (Glance et al., 2013). Previously, technology-driven education has seen many names applied to describe this mode of learning, such as distance education and e-learning (Rudesta & Schoenholtz-Read, 2010).

Typically, two different types of MOOCs have been identified, one being based on existing university courses that embrace the use of videos to deliver content and online assessment ("xMOOCs") (Glance et al., 2013) and the other courses based around online communities and sociocultural constructivist principles called "cMOOCs" (Siemens, 2005).

At their simplest, xMOOCs, the current dominant model, support an instructivist perspective, with the structure of the MOOC focused on the computer-mediated presentation of learning material, with the majority of opportunities for student engagement through automated activities (Toven-Lindsey, Rhoads, & Lozano, 2015). A survey of a range of MOOC stakeholders (n = 106) by Armellini et al. (2016) found that "MOOCs provide good examples of technological innovation but also of highly debatable approaches to pedagogy" (p. 25).

Automated instruction platforms aim to facilitate learning at scale, and they can be personalized through reaction and modification in response to student activity and assessment results. Many xMOOCs adopt a model akin to this

approach, with learners assimilating content in small, incremental steps, with immediate feedback and reinforcement provided through small, automated assessment exercises, such as "missing word" exercises or multiple choice questions. cMOOCs can be seen as the antithesis in many ways, with learners creating content and connecting opportunities for learning as an independent, self-directing community. However, as the development of MOOCs progresses, examples that integrate both cognitive and sociocultural constructivist pedagogic practices with more structured, content-driven learning approaches, forming a *hybrid* MOOC model, have become more common (Bayne & Ross, 2014). Swan, Day, and Bogle (2016) found variation in pedagogic approach in their analysis of 20 MOOCs from across multiple platforms, with increased adoption of collaborative and participatory approaches.

MOOCs provide the potential for the development of new pedagogic approaches within the online learning space, while also facilitating leveraging existing pedagogy for adaptation to the online learning space, particularly when adopted as part of a blended learning approach. There is further opportunity to explore the development of MOOC-specific pedagogy, with MOOCs still tending to adopt traditional distance learning pedagogy (Vassiliadis, Kameas, & Sgouropoulou, 2016).

Falkner et al. (2016) describe the development of a MOOC designed around active and collaborative pedagogic practices, incorporating motivational context through media computation (Rich, Perry, & Guzdial, 2004). They found higher than average participation rates from female learners in comparison to both online and traditionally taught introductory computing courses. While learning outcomes demonstrated effectiveness in building initial skill with fundamental programming concepts in a pattern that repeated through the course, when first designing and using new constructs, students reverted to their more limited usage of previous concepts, indicating a lack of depth of understanding and a need to explore further scaffolding.

Rizzardini and Amado-Salvatierra (2017) describe their MicroMasters MOOC model where the MOOC itself is explicitly positioned as only one of three learning phases, with pre-MOOC and post-MOOC phases designed to facilitate the construction of ongoing learning communities in order to provide opportunities for social learning. Piccioni, Estler, and Meyer (2014) describe their experiences in developing an introductory programming MOOC as a supplementary learning environment for their on-campus course. In this approach, the MOOC was an optional component, with the course retaining its standard approach (i.e., not flipped), with students engaging significantly in both on-campus and MOOC learning activities.

Grover, Pea, and Cooper (2014) present a computational thinking MOOC targeted at middle school, explicitly identifying strategies for promoting active learning within a MOOC environment, including the use of worked examples, code tracing, quizzes, Parsons' problems and hands-on assignment tasks. This approach showed equivalent or slightly better results in learning outcomes compared with previous face-to-face course cohorts. Falkner et al. (2017)

describe the design of a MOOC to support teacher professional development in computing, adopting a sociocultural constructivist perspective, defining active learning through scaffolded, direct application of MOOC content to the professional context, and using a community of practice model of shared repository building.

15.4 Influences on Pedagogic Practices

Within the computing discipline, there are contextual factors that guide and shape the development of pedagogy and influence how it is applied in a particular situation. An important consideration is that there are factors that prevent or constrain application of a pedagogy. For example, changing from a standard lecture or tutorial class to a studio environment may not be possible if there are no suitable teaching spaces available. In some circumstances, the desire to implement a particular pedagogy becomes a strong driver for change. For example, there is evidence to indicate that pedagogy has influenced the design of teaching spaces (and vice versa) (Walker, Brooks, & Baepler, 2011). Some educational development environments developed for learning programming have been designed to facilitate particular pedagogic approaches. For example, Scratch, Alice, and Greenfoot aim to engage and empower students by enabling the development of games, simulations, and stories that can connect to their interests (Utting et al., 2010)

When exploring the pedagogic approach that may be best suited to our purposes, it is important to also understand the context and motivations that most effectively guide learning and the application of pedagogic practice. With the current emphasis on student-centered education, the factors that influence pedagogy are often related to awareness and understanding of students' needs. This places emphasis on practices that address specific learning challenges, attempt to interest and motivate students, and increase access and inclusion.

15.4.1 Addressing Learning Challenges

There are a number of pedagogic approaches that address students' learning challenges specific to the computing discipline. For example, it is understood that explicit awareness of the *notional machine* (du Boulay, 1986) – an abstract representation of the machine that corresponds to program execution in the language being learned – facilitates learning to program and establishing a correct mental model for program execution (this is discussed further below). As another example, we can observe the creation of specific pedagogic practices based around supporting objects-first or objects-last approaches to teaching programming within the object-oriented programming paradigm (e.g., Cooper, Dann, & Pausch, 2003; Kölling & Barnes, 2004; Moritz & Blank, 2005). The desire to prepare students for future work in

industry has motivated the adoption and expansion of pedagogic approaches that build communication, collaboration, and cooperation skills, as well as pedagogic practices explicitly derived from industry practice (e.g., test-driven development).

15.4.2 The Importance of Context and Motivation for Pedagogic Practice

Concerns about low student engagement and poor retention rates in computing courses have been an impetus for pedagogic practices that are tailored to students' interests. For example, teaching programming through the context of games, media computation, or creative coding provides a motivational context that can assist in engaging students and increasing retention rates.

Games are used extensively in computing courses and, in particular, there are many reports of game-based pedagogies in introductory programming courses. When used for a learning activity in a programming course, games can be played or they can be designed and built. A review by Batista et al. (2016) identified a number of reports on the use of game-based construction to teach programming. They found that most studies reported an increase in students' motivation and engagement, as well as other benefits. For example, Leutenegger and Edgington (2007) used a "games-first" approach to teaching introductory programming. Using Flash and ActionScript for game development, they found higher retention and increased interest, and this approach positively influenced both men and women. They argue that games-based curricula can entice students to study computer science and games-based assignments can engage and motivate students, potentially leading to increased learning outcomes. Corral et al. (2014) used a game-based approach with tangible user interfaces (TUIs). In an experimental study (n = 30) that compared their innovative approach with a traditional approach, they found that students in the game-based TUI approach showed higher interest and achieved higher grades than students in the traditional course. With all of the changes needed to incorporate games into a course, Batista et al. (2016) argue for the need and propose a framework for an effective game-based pedagogy. While there is extensive evidence to support games as a motivating context, there is a lack of critical research to support increased learning outcomes (Battistella & Gresse von Wangenheim, 2016).

Another game-related pedagogic approach is gamification, where game design elements such as leaderboards, badges, and constant feedback are incorporated into instruction. Pirker, Riffnaller-Schiefer, and Gütl (2014) used gamification in their pedagogic approach in an introductory programming class. An end-of-semester survey (n = 21) showed that students found the course more motivating than other courses they had taken and identified a higher degree of interactivity. While numerous studies exist applying gamification and badges as a motivational approach, there is evidence that these forms of extrinsic motivation may be problematic for long-term learning gain and broader learning motivation

(Falkner & Falkner, 2014). There is a need for further research to isolate the specific benefits of gamification and to ascertain any lasting impact.

Media computation is a pedagogic approach for teaching introductory programming whereby manipulation of media, such as digital images and sounds, is used as a context to teach programming concepts. The purpose of using media computation is to make the learning of programming engaging and relevant to students, resulting in increased retention and learning outcomes. Guzdial (2013), in a review of ten years of running media computation courses, found evidence in support of increases in course retention, but not in learning gains. A particular goal of Guzdial's use of media computation was to increase the engagement and success of female students. From interview studies, Guzdial reports that female students found the content useful and the course more motivating than traditional courses.

Another pedagogic approach designed to interest and motivate is where students learn to program using physical devices. Some education technologies have been developed for this purpose, such as Lego Mindstorms (Lui et al., 2010; Wakeling, 2008). Programming physical devices gives students real, hands-on experience with immediate feedback, creating an engaging learning experience; however, they also pose distinct challenges through the cognitive load of understanding the physical device itself (Desportes et al., 2016). Summet et al. (2009) describe their introductory programming class where each student has their own robot, giving them the freedom to explore these devices both during and out of class. Blank (2006) proposes that giving each student their own robot personalizes the learning experience, helping to attract more students and a diversity of students to computer science.

15.4.3 Accessibility and Inclusivity

Computing programs are characterized by low diversity within the student population. Concerns about equity and diversity issues such as low numbers of female students have inspired pedagogic approaches that are accessible and inclusive. For example, giving students experiences that demonstrate the diversity of computing applications and emphasize their social relevance has been argued to promote the accessibility and inclusivity of a computing course (Buckley, Nordlinger, & Subramanian, 2008; Fisher & Margolis, 2002; Khan & Luxton-Reilly, 2016).

A longitudinal study by Fisher and Margolis (2002) involving over 230 interviews of computer science students found that the stereotypical view of the "geek" computing student led to an alienating culture that reduced confidence, interest, and ultimately success rates, and this negative impact was greater on females and minority groups. To remedy this, Margolis and Fisher proposed a pedagogic approach that puts computing in a social context and shows students that there are multiple ways to be a computer scientist.

Aligned to this view, Pulimood and Wolz (2008) propose that equity in computer science programs can be achieved through changing the classroom

pedagogy. They describe their implementations of authentic learning environments where students work in heterogeneous groups solving real problems. They argue that collaborative problem-solving in a gender-neutral, culturally and ethnically diverse, and multidisciplinary groups will make computer science more accessible to females and minority groups. Pulimood and Wolz also claim that this approach better prepares students for the workforce.

Rubio et al. (2015) contend that "if we want to include more women in computing, the pedagogy of introductory programming courses needs to change" (p. 410). They designed and implemented modules using physical computing activities with the Arduino microcontroller board. In an experimental study (n = 38) to determine any differences in the perceptions of learning to program and learning outcomes, they found that, with traditional teaching methods, women perceived programming to be harder than men and they indicated less desire to continue with programming in their study or profession. However; there was no difference with the group that learned to program using the Arduino device.

From a similar motivation of accessibility but with a different approach, Brady et al. (2017) engage students' interest through projects with contexts that illustrate the social and collaborative aspects of computer science. They use physical computing activities and participatory simulations to prepare students for the diverse domain of computing. Brady et al. describe the implementation of this pedagogic approach in a Computational Thinking for Girls (CT4G) project and propose that showing students the social and diverse aspects of computing can be a powerful tool to engage underrepresented groups.

Medel and Pornaghshband (2017) use examples of teaching material from different computer science courses to demonstrate how a pedagogic approach can introduce a gender bias through particular representations of gender, stereotypical imagery, and male-centered language. Medel and Pornaghshband offer suggestions for reducing or eliminating this bias to maximize gender inclusion.

15.4.4 The Notional Machine and Supporting Pedagogic Practices

The act of learning programming involves concurrently developing skill and knowledge in several related areas: planning, design, programming language structure, and also an understanding of how programs are to be executed. Students must learn both how to craft programs to achieve a fixed output as well as comprehending what an existing program will do. Knowledge of the *notional machine*, as defined by du Boulay (1986), consists of "the general properties of the machine that one is learning to control," as defined by the semantics of the programming language in use, and is designed to facilitate learning of programming by making program execution explicit as part of the learning process. The notional machine formed an important concept with a strong influence on pedagogic development within computer science and in informing a significant amount of computer science education research (see Chapter 12 for further description).

Pedagogic practices have been introduced in order to support the notional machine as an abstraction of programming execution designed to expose those elements of the execution process that will aid understanding, such as memory usage and instruction sequencing, without exposing those details that are unnecessary to form the appropriate mental model of programming semantics. The level of detail required in our notional machine varies by the learning context and the depth and detail of the required mental model to be established (Sorva, 2013); similarly, our notional machine will vary depending on the programming language in use. From this perspective, a notional machine is an abstract, idealized representation of aspects of runtime execution for a specific programming language and context that are relevant to supporting the understanding of how a program is executed.

Without an appropriate mental model of a notional machine, it is considered that student knowledge of the programming process and processing languages is "fragile" (Perkins, Schwartz, & Simmons, 1990) and that challenges faced by novices in learning to program are frequently due to their mental model of the notional machine being "inadequate" (Smith & Webb, 1995). Without making the notional machine explicit in our learning processes, we risk students forming incomplete or conflicting mental models that are littered with misconceptions and often informed by guesswork. Indeed, when forming initial mental models, beginners tend to attribute human reasoning to the computer, incorporating assumptions about the semantics implied by natural language analogies of programming language terms (du Boulay, O'Shea, & Monk, 1981), the semantics of variables names, or the positioning of statements within the overall code (Ragonis & Ben-Ari, 2005) (see Chapter 13 for a related discussion).

Research designed to support notional machine mental model construction has introduced both pedagogic approaches (e.g., comprehension-first and design-based pedagogy) as well as instructional approaches, such as the construction of program writing and visualization tools to facilitate exposing the notional machine as part of the learning process.

Visualization in particular has developed as a common instructional device for supporting comprehension-first or comprehension-based pedagogy. Sorva, Karavirta, and Malmi (2013) present a comprehensive review of generic program visualization systems used for introductory programming education, exploring visualization as a technique for establishing a notional machine, as well as more broadly to support the development of conceptions of the dynamic nature of program execution, addressing common misconceptions, and supporting the skill of tracing.

Visualizations, be they manual visualization exercises or sophisticated, automated tools, have long been proposed as a mechanism for establishing a mental model of a notional machine, from early work by Mayer (1975, 1976) through to Cunningham et al.'s (2017) recent replication study of the use of sketching and sketching strategies to illustrate understanding, identifying benefits in distributing cognition and managing cognitive load. A constructivist

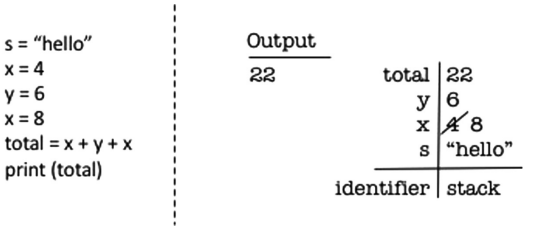

Figure 15.2 *Memory visualization exploring primitive data types and the correspondence between variable name and value (adapted from Dragon & Dickson, 2016).*

perspective encourages students to actively engage with visualizations through active and collaborative learning activities instead of solely observing visualization operation (Hundhausen, Douglas, & Stasko, 2002). There are many examples of visualization activities and tools, and we would advise the reader to review Sorva et al. (2013) and Hundhausen et al. (2002) for more detailed discussions. However, we will present a small number of examples here to highlight the applicability of these approaches and current practice.

Hertz and Jump (2013) and Dragon and Dickson (2016) present hand-drawn approaches to tracing memory usage within programs to help establish a mental model of the notional machine and facilitate understanding of program execution. Both approaches describe interactive student activities where students work through a series of tracing exercises using established templates for memory visualization. Dragon and Dickson present multiple examples, crossing languages and concepts, illustrating the development of the notional machine and its increasing complexity through increasing detail and sophistication of their visualization techniques as students' understanding and experience develop, moving from a simple stack visualization (see Figure 15.2) through to examples supporting visualization of pointers, explicit vs. implicit allocation, and deallocation (see Figure 15.3).

Recent work in the context of visualization tools include Sorva and Sirkiä's (2010) work within the UUhistle environment (a highly interactive visualization environment representing memory as abstract graphics) that introduces the idea of visual program simulation as an explicit active learning activity that requires learners to control the progression of the simulation – in a sense, acting as the "computer." Sirkliä and Sorva (2015) extend their efforts in this domain with more recent work in the development of a new program visualization framework that supports educators in integrating program visualizations into their teaching context, including annotations with text, audio, and interactive activities.

Another recent development is Guo's (2013) web-based Python simulation and visualization environment that allows learners to move backward and forward through an execution, visualizing stack frames and variables, heap object contents, and memory. While earlier work has long provided similar technical support for this degree of program simulation and visualization (Findler et al, 2002), an interesting aspect of this tool is the ability to generate a URL for a

```
#include <iostream>
using namespace std;
int main () {
   int stackX = 5;
   int* stackPtr = &stackX;
   int* heapPtr = new int;
   *stackPtr = 10;
   *heapPtr = 20;
   cout << "\n on stack: " << stackX;
   cout << "\m on heap: " << *heapPtr;
   delete heapPtr;
   heapPtr = nullptr;
   return 0;
}
```

Output

```
on stack: 10
on heap: 20
```

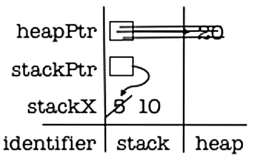

Figure 15.3 *Memory visualization of a C++ program facilitating understanding of memory addressing, allocation, and deallocation (adapted from Dragon & Dickson, 2016).*

specific point in the execution visualization, which can be shared among learners to aid collaboration and discussion.

More recent work has explored the explicit structuring of much of this pedagogic basis in the form of an explicit comprehension-first pedagogy (Nelson, Xie, & Ko, 2017), where programming semantics are taught prior to the learning of how to program. In this example of comprehension-first pedagogy, students learn to trace and understand the execution of a program prior to learning program development and through the execution of a notional machine.

15.5 Challenges and Future Directions

Computing educators have many options for their pedagogic practices, and there is much evidence of innovative teaching techniques, tools, and resources in computing education. Much of the recent effort has been in response to a focus on student-centered and active learning and has been largely facilitated by advancements in computing and information technology. Despite these efforts, however, computing educators face a number of challenges. A fundamental issue is an apparent mismatch between the pedagogy used and the way students choose to learn. This is widely evidenced by changes in the way that students engage with traditional learning resources, including teaching spaces (Kandiko & Mawer, 2013; Morgan et al., 2017). These issues are compounded by changing student populations. As enrollments increase in some areas of the world (Computing Research Association, 2017; CSER, 2017), we are seeing increasing concern over maintaining diversity and quality within our learning environments. The Generation CS report (Computing Research Association, 2017) presents the impact of growth across the USA and Canada, identifying challenges in providing appropriate teaching spaces for preferred pedagogic practices and changes in the pedagogic practices employed. This report also

indicates changes in our cohorts, with an increase in non-majors enrolling in computer science courses and variation in representation of minority groups. Patitsas, Craig, and Easterbrook (2016) argue that a number of policy changes made in response to growing enrollments have a direct negative impact on diversity, presenting a worrying trend for our discipline over coming years and a need for further research on the relationship between policy development and pedagogic practice.

There are also opportunities within the computing discipline for creative pedagogies, with technology often an inspiration for change and a key component of a teaching innovation. Sharples et al. (2016), in their Open University Innovation Pedagogy 2016 Report, identify ten emerging innovations that are predicted to cause major changes in pedagogic approach and practice: learning through social media, productive failure, teachback, design thinking, learning from the crowd, learning from video games, formative analytics, learning for the future, translanguaging, and blockchain for the future. These innovations offer a lens through which to view the evolution of computer science pedagogy over the coming years and to motivate future research into the specific impact upon computer science education research and pedagogy.

Learning through social media addresses increased usage of social media platforms (both specific to learning and in general) on promoting different ways of engaging with learners and different conceptualizations of the role of the teacher. Early work within the computer science field includes Vickers et al. (2014) and Vozniuk, Holzer, and Gillet (2014). The increased usage of social media platforms, along with other online learning spaces, presents a challenge for educators and learners in developing pedagogic approaches that support learners in moving seamlessly between environments while maintaining scaffolding and consistency of learning.

Productive failure encourages students to tackle challenging problems with potential for failure prior to receiving direct instruction. While associated in structure with problem-based learning approaches, productive failure has a specific emphasis on the use of failure as a pivotal point in the learning process (Kapur, 2008; Schneider & Blikstein, 2016). Lui et al. (2017) explore the use of productive failure in their work within inclusive pedagogy design using e-textiles; however, this is one of few such works in computer science education research to date.

Teachback, where learners explain back to the teacher what they have learned and what they think they know, helps the learner to reflect on and conceptualize gaps in their learning. Aligned with CSP approaches and building on early pedagogic development (Pask, 1976), teachback has a specific focus on being taught then "teaching back" to deepen knowledge, increase reflection, and increase transparency of learning.

There is an opportunity to exploit artificial intelligence techniques in the creation of new pedagogic practices associated with teachback, particularly in online environments that may support growth and scale requirements. Rudman (2002) constructs an automated approach to analyzing discourse

(captured digitally) in order to identify associated resource suggestions. There are further opportunities to expand upon existing work in applying natural language processing techniques in the exploration of automated concept map development (Atapattu, Falkner, & Falkner, 2017), topic modeling, and discourse analysis (Atapattu & Falkner, 2016) to assist learners in visualizing and connecting their explanations and also to assist in contrasting explanations and conceptualizations of learning.

Design thinking is a structured approach to problem-solving that supports episodic phases of creativity and critical thinking, identifying possibilities and constraints before moving on to analysis and construction – the "solving" process itself. While this is a long-standing pedagogic approach (Lawson, 2005), this approach has gained recent interest as a means for building innovation and creativity skills (Koh et al., 2015).

Learning from the crowd uses crowdsourced information and platforms as the basis for learning, establishing pedagogic practices that use and engage with crowdsourcing platforms and approaches to solve problems and gather information. Again, there are opportunities here for computer science education research in terms of exploring learning at scale through crowdsourced participation in educational studies.

Learning through video games builds on an extensive history of games within learning – video game pedagogies that exploit game characteristics that trigger and sustain learning. There has been extensive work exploring the development of video games within computing education (Battistella & Gresse von Wangenheim, 2016), including digital systems (Srinivasan, Butler-Purry, & Pedersen, 2008) and recursion (Lee et al., 2014), but with the majority focused on software engineering education. However, in their systematic literature review, Battistella and Gresse von Wangenheim (2016) identify a lack of integration into the learning context, with a need for further research in establishing connection with increased learning outcomes and designing game contexts that support learning.

Formative analytics expands upon the use of learning analytics to report on learning (the more traditional view of learning analytics) toward the development of analytics directly informing and assisting the learner (Higher Education Commission, 2016). As discussed further below, this represents a valuable area for computer science education research, one that can influence the refinement and development of new pedagogic approaches. Formative analytics that support personalized learning with awareness of stereotype threat and associated equity concerns is one potential strategy to explore here (see Chapter 16 for further discussion).

The growth of learning analytics, online learning, and learning at scale has provided computer science educators with new data sources and experimental techniques to explore how our students learn. The increased availability of detailed learner behavior and interactions in MOOC environments, as well as other large-scale learning platforms, provides a new opportunity to analyze and understand fine-grained learner behavior and the impact of pedagogic changes

on learner outcomes (Milligan, 2015; Milligan et al., 2016). There is considerable opportunity for expanded computer science education research in this space, including the identification of relevant data points associated with measuring learning, pedagogic approaches that leverage formative analytics, the establishment of frameworks for analyzing and reporting on learning outcomes, assessment of pedagogy, and the development of analytics frameworks supporting rigorous comparison of pedagogic approaches.

Learning for the future encompasses a range of pedagogic approaches that support autonomy, resourcefulness, critical thinking skills, and interpersonal skills, building on existing pedagogic approaches and practices, including studio-based learning, project-based learning, and personalized learning assisted by learning analytics.

Translanguaging recognizes the global nature of future learning environments and the subsequent need for pedagogic practices and environments that support multi-language development and encourage diversity in language and cultural context (Creese & Blackledge, 2010).

The final innovation, *blockchain for learning*, defines new ways of storing, validating, and trading educational qualifications and evidence for learning. Pedagogic approaches, building on early attempts at gamification, can be established here, providing new opportunities for understanding and exploring the benefits of gamification. Further, these mechanisms provide a means for establishing reputation (as a learner or as a professional) and provide new means for engaging with social media and crowdsourced pedagogies (as described above).

Many of the innovation opportunities identified by Sharples et al. (2016) are driven by increased technology usage, accessibility, and integration with education, whether it be increased pervasiveness of technology platforms, such as social media and blockchain, or increased opportunity to influence and use data on learner behavior and outcomes. Increased technology integration and evolution of our physical teaching spaces also afford opportunities for the evolution of pedagogic practice, seen already through increases in physical computing and studio-based approaches. How we develop pedagogic approaches and practices that negotiate the abundance of online learning spaces, mediated through technology-rich physical environments, remains an important question for future research.

15.6 Conclusion

The variety of pedagogical approaches that are used in computing education take different forms that we have broadly categorized as active learning, collaborative learning, cooperative learning, CSP, blended learning, and MOOCs. Associated with these approaches are many different and creative teaching techniques, often supported by innovative tools and resources. The development of these has been motivated by a range of contextual and external factors.

We foresee that pedagogic approaches will continue to evolve in response to the changing situation in computing programs and their student populations, making this area a fertile ground for future computing education research.

References

Abad, C. L. (2008). Learning through creating learning objects: Experiences with a class project in a distributed systems course. In *13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)* (pp. 255–259). New York: ACM.

Armellini, A., & Padilla Rodriguez, B. C. (2016). Are Massive Open Online Courses (MOOCs) pedagogically innovative? In *Journal of Interactive Online Learning*, 14(1), 17–28.

Aronson, E., Blaney, N., Stephen, C., Sikes, J., & Snapp, M. (1978). *The Jigsaw Classroom*. Beverly Hills, CA: Sage.

Atkins, M. J. (1993). Theories of learning and multimedia: An overview. *Research Papers in Education*, 8(2), 251–271.

Atapattu, T., & Falkner, K. (2016). A framework for topic generation and labeling from MOOC discussions. In *3rd ACM Conference on Learning@Scale (L@S'2016)* (pp. 201–204). New York, NY: ACM.

Atapattu, T., Falkner, K., & Falkner, N. (2017). A comprehensive text analysis of lecture slides to generate concept maps. *Computers & Education*, 115, 96–113.

Barrows. H. S. (1986). A taxonomy of problem-based learning methods. *Medical Education*, 20, 481–486.

Batista, A. L. F., Connolly, T., & Angotti, J. A. P. (2016). A framework for games-based construction learning: A text-based programming languages approach. In *European Conference on Games Based Learning* (pp. 815–823). Reading, UK: Academic Conferences International Ltd.

Battistella, P., & Gresse von Wangenheim, C. (2016). Games for teaching computing in higher education – A systematic review. *IEEE Technology and Engineering Education*, 1(3), 8–30.

Bayne, S., & Ross, J. (2014). *The Pedagogy of the Massive Open Online Course: The UK view*. York, UK: The Higher Education Academy.

Beck, K. (2002). *Test-Driven Development: By Example*. Boston, MA: Addison-Wesley.

Beck, L., & Chizhik, A. (2013). Cooperative learning instructional methods for CS1: Design, implementation, and evaluation. *Transactions on Computing Education (TOCE)*, 13(3), 10.

Ben-Ari, M. (1998). Constructivism in computer science education. In D. Joyce, D. & J. Impagliazzo (Eds.), *29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98)* (pp. 257–261). New York: ACM.

Bishop, J. L., & Verleger, M. A. (2013). The flipped classroom: A survey of the research. *ASEE National Conference*, 30(9), 1–18.

Blanc, R., DeBuhr, L., & Martin, D. (1983). Breaking the attrition cycle: The effects of supplemental instruction on undergraduate performance and attrition. *The Journal of Higher Education*, 54(1), 80–90.

Blank, D. (2006). Robots make computer science personal. *Communications of the ACM*, 9(12), 5–27.

Brady, C., Orton, K., Weintrop, D., Anton, G., Rodriguez, S., & Wilensky, U. (2017). All roads lead to computing: making, participatory simulations, and social computing as pathways to computer science. *IEEE Transactions on Education*, 60(1), 59–66.

Buckley, M., Nordlinger, J., & Subramanian, D. (2008). Socially relevant computing. In *39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)* (pp. 347–351). New York: ACM.

Buffardi, K., & Edwards, S. H. (2012). Impacts of teaching test-driven development to novice programmers. *International Journal of Information and Computer Science*, 1(6), 135–143.

Carbone, A., & Sheard, J. (2002). A studio-based teaching and learning model in IT: What do first year students think? In *7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '02)* (pp. 213–217) New York: ACM.

Ching, E., Chen, C. T., Chou, C. Y., Deng, Y. C., & Chan, T. W. (2005). A pilot study of computer supported learning by constructing instruction notes and peer expository instruction. In *Conference on Computer Support for Collaborative Learning: Learning 2005: The Next 10 Years! (CSCL)* (pp. 63–67). Taipei, Taiwan: International Society of the Learning Sciences.

Clarke, D., Clear, T., Fisler, K., Hauswirth, M., Krishnamurthi, S., Politz, J. G., Tirronen, V., & Wrigstad, T. (2014). In-flow peer review. In A. Clear & R. Lister (Eds.), *Working Group Reports of the 2014 Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR '14)* (pp. 59–79). New York: ACM.

Collis, B., & Moonen, J. (2005). *An On-Going Journey: Technology as a Learning Workbench*. Enschede, The Netherlands: University of Twente.

Computing Research Association (2017). Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006. Retrieved from https://cra.org/data/Generation-CS/

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in introductory computer science. In *34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03)* (pp. 191–195). New York: ACM.

Corral, J. M. R., Balcells, A. C., Estévez, A. M., Moreno, G. J., & Ramos, M. J. F. (2014). A game-based approach to the teaching of object-oriented programming languages. *Computers & Education*, 73, 83–92.

Creese, A., & Blackledge, A. (2010). Translanguaging in the bilingual classroom: A pedagogy for learning and teaching? *The Modern Language Journal*, 94(1), 103–115.

Crescenzi, P., & Nocentini, C. (2007). Fully integrating algorithm visualization into a CS2 course: A two-year experience. In *12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '07)* (pp. 296–300). New York: ACM.

CSER (2017). A look at IT and Engineering Enrolments in Australia – Updated. Retrieved from: https://blogs.adelaide.edu.au/cser/2017/02/15/a-look-at-it-and-engineering-enrolments-in-australia-updated/

Cunningham, K., Blanchard, S., Ericson, B., & Guzdial, G. (2017). Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In *2017 ACM Conference on International Computing Education Research (ICER '17)* (pp. 164–172). New York: ACM.

Day, J. A., & Foley, J. D. (2006). Evaluating a web lecture intervention in a human–computer interaction course. *IEEE Transactions on Education*, 49(4), 420–431.

Denny, P., Luxton-Reilly, A., & Hamer, J. (2008). The PeerWise system of student contributed assessment questions. In Simon & M. Hamilton (Eds.), *10th conference on Australasian computing education – Volume 78 (ACE '08)*, Vol. 78 (pp. 69–74). Darlinghurst, Australia: Australian Computer Society, Inc.

DesPortes, K., Anupam, A., Pathak, N., & DiSalvo, B. (2016). BitBlox: A redesign of the breadboard. In *15th International Conference on Interaction Design and Children (IDC '16)* (pp. 255–261). New York: ACM.

Dragon, T., & Dickson, P. E. (2016). Memory diagrams: A consistent approach across concepts and languages. In *47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)* (pp. 546–551). New York: ACM.

du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57–73.

du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. *International Journal of Man-Machine Studies*, 14(3), 237–249.

Duffy, T. M., & Cunningham, D. J. (1996). Constructivism: Implications for the design and delivery of instruction. In D. H. Jonassen (Ed.), *Handbook of Research for Educational Communications and Technology* (pp. 170–198). New York: Macmillan.

Earl, S. E. (1986). Staff and peer assessment – Measuring an individual's contribution to group performance. *Assessment & Evaluation in Higher Education*, 11(1), 60–69.

Ebert, M., & Ring, M. (2016). A presentation framework for programming in programing lectures. In *Global Engineering Education Conference* (EDUCON '16) (pp. 369–374). New York, NY: IEEE.

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04)* (pp. 26–30). New York: ACM.

Elgort, I., Smith, A., & Toland, J. (2008). Is Wiki an effective platform for group course work? *Australasian Journal of Educational Technology*, 24(2), 195–210.

Ernest, E. (1995). The one and the many. In L. Steffe & J. Gale (Eds.), *Constructivism in Education* (pp. 459–486). New York: Lawrence Erlbaum.

Ewing, J. M., Dowling, J. D., & Coutts, N. (1998). Learning using the World Wide Web: A collaborative learning event. *Journal of Educational Multimedia and Hypermedia*, 8(1), 3–22.

Falkner, N. J. G., & Falkner, K. (2014). "Whither, badges?" or "wither, badges!": A metastudy of badges in computer science education to clarify effects, significance and influence. In *14th Koli Calling International Conference on Computing Education Research (Koli Calling '14)* (pp. 127–135). New York: ACM.

Falkner, K., Falkner, N., Szabo, C., & Vivian, R. (2016). Applying validated pedagogy to MOOCs: An introductory programming course with media computation. In *2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 326–331). New York, NY: ACM.

Falkner, K., & Palmer, E. (2009). Developing authentic problem solving skills in introductory computing classes. In *40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)* (pp. 4–8). New York: ACM.

Falkner, K., Vivian, R., Falkner, N., & Williams, S. (2017). Reflecting on three offerings of a community-centric MOOC for k-6 computer science teachers. In *2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)* (pp. 195–200). New York: ACM.

Findler, R. B., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P., & Felleisen, M. (2002). DrScheme: A programming environment for Scheme. *Journal of. Functional Programming*, 12(2), 159–182.

Firmin, S., Sheard, J., Carbone, A., & Hurst, J. (2012). An exploration of factors influencing tertiary IT educators' pedagogies. In M. de Raadt & A. Carbone (Eds.), *14th Australasian Computing Education Conference – Volume 123 (ACE '12)*, Vol. 123 (pp. 157–166). Darlinghurst, Australia: Australian Computer Society, Inc.

Fisher, A., & Margolis, J. (2002). Unlocking the clubhouse: The Carnegie Mellon experience. *SIGCSE Bulletin*, 34(2), 79–83.

Geer, U., & Rudge, D. (2002). A review of research on constructivist-based strategies for large lecture science classes. *Electronic Journal of Science Education*, 7(2). Retrieved from www.scholarlyexchange.org/ojs/index.php/EJSE/article/view/7701

Gehringer, E. F., & Miller, C. S. (2009). Student-generated active-learning exercises. In *40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)* (pp. 81–85). New York: ACM.

Giannakos, M. N., Krogstie, J., & Chrisochoides, N. (2014). Reviewing the flipped classroom research: Reflections for computer science education. In E. Barendsen & V. Dagiené (Eds.), *Computer Science Education Research Conference (CSERC '14)* (pp. 23–29). New York: ACM.

Glance D., Forsey M., & Riley M. (2013). The pedagogical foundations of massive open online courses. *First Monday,* 18(5). Retrieved from http://firstmonday.org/ojs/index.php/fm/article/view/4350/3673

Grover, S., Pea, R., & Cooper, S. (2014). Promoting active learning & leveraging dashboards for curriculum assessment in an OpenEdX introductory CS course for middle school. In *1st ACM Conference on Learning @ Scale Conference (L@S '14)* (pp. 205–206). New York: ACM.

Guo, P. J. (2013). Online python tutor: Embeddable web-based program visualization for cs education. In *44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)* (pp. 579–584). New York: ACM.

Guzdial, M. (2013). Exploring hypotheses about media computation. In *9th Annual International ACM Conference on International Computing Education Research (ICER '13)* (pp. 19–26). New York: ACM.

Hamer, J., Cutts, Q., Jackova, J., Luxton-Reilly, A., McCartney, R., Purchase, H., Riedesel, C., Saeli, M., Sanders, K., & Sheard, J. (2008). Contributing student pedagogy. *SIGCSE Bulletin*, 40(4), 194–212.

Hamer, J., Luxton-Reilly, A., Purchase, H. C., & Sheard, J. (2011). Tools for "contributing student learning". *ACM Inroads*, 2(2), 78–91.

Hativa, N. (2000). Active learning during lectures. In *Teaching for Effective Learning in Higher Education* (pp. 87–110). Dordrecht, The Netherlands: Springer.

Hertz, M., & Jump, M. (2013). Trace-based teaching in early programming courses. In *44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)* (pp. 561–566). New York: ACM.

Higher Education Commission (2016). *From Bricks to Clicks: The Potential of Data and Analytics in Higher Education*. London, UK: Higher Education Commission.

Horton, D., & Craig, M. (2015). Drop, fail, pass, continue: Persistence in CS1 and beyond in traditional and inverted delivery. In *46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)* (pp. 235–240). New York: ACM.

Hundhausen, C. D., Agrawal, A., & Agarwal, P. (2013). Talking about code: Integrating pedagogical code reviews into early computing courses. *ACM Transactions on Computing Education (TOCE)*. 13(3), 14.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3), 259–290.

Hundhausen, C. D., Narayanan, N. H., & Crosby, M. E. (2008). Exploring studio-based instructional models for computing education. In *39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)* (pp. 392–396). New York: ACM.

Jonas, M. (2013). Group note taking in Mediawiki, a collaborative approach. In *14th Annual ACM SIGITE Conference on Information Technology Education (SIGITE '13)* (pp. 131–132). New York: ACM.

Jonassen, D. H. (1991). Objectivism versus constructivism: Do we need a new philosophical paradigm? *Educational Technology Research & Development*, 39(3), 5–14.

Kaczmarczyk, L., Boutell, M., & Last, M. (2007). Challenging the advanced first-year student's learning process through student presentations. In *3rd International Workshop on Computing Education Research (ICER '07)* (pp. 17–26). New York: ACM.

Kandiko, C. B., & Mawer, M. (2013). *Student Expectations and Perceptions of Higher Education*. London, UK: King's Learning Institute.

Kapur, M. (2008). Productive failure. *Cognition and Instruction*, 26(3), 379–424.

Khan, N. Z., & Luxton-Reilly, A. (2016). Is computing for social good the solution to closing the gender gap in computer science? In *Australasian Computer Science Week Multiconference (ACSW '16)* (p. 17). New York: ACM.

Koh, J. H. L., Chai, C. S., Wong, B., & Hong, H.-Y. (2015). *Design Thinking for Education*. Singapore: Springer.

Kölling, M., & Barnes, D. J. (2004). Enhancing apprentice-based learning of Java. In *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04)* (pp. 286–290). New York: ACM.

Kurtz, B. L., Fenwick, J. B., Tashakkori, R., Esmail, A., & Tate, S. R. (2014). Active learning during lecture using tablets. In *45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)* (pp. 121–126). New York: ACM.

Lacher, L. L., & Lewis, M. C. (2015). The effectiveness of video quizzes in a flipped class. In *46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)* (pp. 224–228). New York: ACM.

Lawson, B. (2005). *How Designers Think: The Design Process Demystified*, 4th edn. London, UK: The Architectural Press.

Lee, E., Shan, V., Beth, B., & Lin, C. (2014). A structured approach to teaching recursion using cargo-bot. In *10th Annual Conference on International Computing Education Research (ICER '14)* (pp. 59–66). New York: ACM.

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. In *38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)* (pp. 115–118). New York: ACM.

Lui, D., Anderson, E., Kafai, Y. B., & Jayathirtha, G. (2017). Learning by fixing and designing problems: A reconstruction kit for debugging e-textiles. In *7th Annual Conference on Creativity and Fabrication in Education (FabLearn '17)* (p. 6). New York: ACM.

Lui, A. K., Ng, S. C., Cheung, Y. H. Y., & Gurung, P. (2010). Facilitating independent learning with Lego Mindstorms robots. *ACM Inroads*, 1(4), 49–53.

Lockwood, K., & Esselstein, R. (2013). The inverted classroom and the CS curriculum. In *44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)* (pp. 113–118). New York: ACM.

Lowyck, J., & Elen, J. (1993). Transitions in the theoretical foundation of instructional design. In T. M. Duffy, J. Lowyck, & D. H. Jonassen (Eds.), *Designing Environments for Constructive Learning* (pp. 213–229). Berlin, Germany: Springer-Verlag.

Lutteroth, C., & Luxton-Reilly, A. (2008). Flexible learning in CS2: A case study. In *21st Annual Conference of the National Advisory Committee on Computing Qualifications* (pp. 77–83). New Zealand: Computing and Information Technology Research and Education New Zealand (CITRENZ).

Luxton-Reilly, A., Bertinshaw, D., Denny, P., Plimmer, B., & Sheehan, R. (2012). The impact of question generation activities on performance. In *43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)* (pp. 391–396). New York: ACM.

Luxton-Reilly, A., Plimmer, B., & Sheehan, R. (2010). StudySieve: A tool that supports constructive evaluation for free-response questions. In *11th International Conference of the NZ Chapter of the ACM Special Interest Group on Human-Computer Interaction (CHINZ '10)* (pp. 65–68). New York: ACM.

Maher, M. L., Latulipe, C., Lipford, H., & Rorrer, A. (2015). Flipped classroom strategies for CS education. In *46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)* (pp. 218–223). New York: ACM.

Mayer, R. E. (1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. *Journal of Educational Psychology*, 67(6), 725–734.

Mayer, R. E. (1976). Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order. *Journal of Educational Psychology*, 68(2), 143–150.

Medel, P., & Pournaghshband, V. (2017). Eliminating gender bias in computer science education materials. In *2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)* (pp. 411–416). New York: ACM.

Milligan, S. (2015). Crowd-sourced learning in MOOCs: learning analytics meets measurement theory. In *5th International Conference on Learning Analytics & Knowledge (LAK '15)* (pp. 151–155). New York: ACM.

Milligan, S., He, J., Bailey, J., Zhang, R., & Rubinstein, B. I. P. (2016). Validity: A framework for cross-disciplinary collaboration in mining indicators of learning from MOOC forums. In *6th International Conference on Learning Analytics & Knowledge (LAK '16)* (pp. 546–547). New York: ACM.

Moog, R. S. & Spencer, J. N. (Eds.) (2008). *Process-Oriented Guided-Inquiry Learning (POGIL)*. Washington, DC: American Chemical Society.

Morgan, M., Butler, M., Sinclair, J., Cross, G., Fraser, J., Jackova, J., & Thota, N. (2017). Understanding international benchmarks on student engagement: Awareness, research alignment and response from a computer science perspective. In *2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)* (pp. 383–384). New York: ACM.

Moritz, S. H., & Blank, G. D. (2005). A design-first curriculum for teaching Java in a CS1 course. *SIGCSE Bulletin*, 37(2), 89–93.

Nelson, G. L., Xie, B., & Ko, A. J. (2017). Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1. In *2017 ACM Conference on International Computing Education Research (ICER '17)* (pp. 2–11). New York: ACM.

Nortcliffe, A. (2005). Student-driven module: promoting independent learning. *International Journal of Electrical Engineering Education*, 42(3), 247–512.

Papert, S. (1993). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. In D. Tolhurst & S. Mann (Eds.), *8th Australasian Conference on Computing Education – Volume 52 (ACE '06)* (pp. 157–163). Darlinghurst, Australia: Australian Computer Society, Inc.

Pask, G. (1976). *Conversation Theory, Applications in Education and Epistemology*. Amsterdam, The Netherlands: Elsevier.

Patitsas, E., Craig, M., & Easterbrook, S. (2016). How CS departments are managing the enrolment boom: Troubling implications for diversity. In *Research on Equity and Sustained Participation in Engineering, Computing and Technology (RESPECT)* (pp. 1–2). New York, NY: IEEE.

Paul, J. (2016). Test-driven approach in programming pedagogy. *Journal of Computing Sciences in Colleges*, 32(2), 53–60.

Perkins, D. N., Schwartz, S., & Simmons, R. (1990). Instructional strategies for the problems of novice programmers. In R. E. Meyer (Ed.), *Teaching and Learning Computer Programming: Multiple Research Perspectives* (pp. 153–178). Mahwah, NJ: Lawrence Erlbaum.

Phillips, D., & Soltis, J. F. (2015). *Perspectives on Learning*, 5th edn. New York: Teachers College Press.

Piaget, J. (1972). *Psychology and Epistemology: Towards a Theory of Knowledge* (Vol. 105). London, UK: Penguin Books Ltd.

Piccioni, M., Estler, C., & Meyer, B. (2014). SPOC-supported introduction to programming. In *19th Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)* (pp. 3–8). New York: ACM.

Pirker, J., Riffnaller-Schiefer, M., & Gütl, C. (2014). Motivational active learning: engaging university students in computer science education. In *2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)* (pp. 297–302). New York: ACM.

Politz, J. G., Krishnamurthi, S., & Fisler, K. (2014). In-flow peer-review of tests in test-first programming. In *10th Annual Conference on International Computing Education Research (ICER '14)* (pp. 11–18). New York: ACM.

Pollock, L., & Harvey, T. (2011). Combining multiple pedagogies to boost learning and enthusiasm. In *16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)* (pp. 258–262). New York: ACM.

Porter, L., & Simon, B. (2013). Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)* (pp. 165–170). New York: ACM.

Portillo, J. A. P., & Campos, P. G. (2009). The jigsaw technique: Experiences teaching analysis class diagrams. In *Mexican International Conference on Computer Science* (pp. 289–293). New York: IEEE Press.

Pulimood, S. M., & Wolz, U. (2008). Problem solving in community: A necessary shift in cs pedagogy. *ACM SIGCSE Bulletin*, 40(1), 210–214.

Purchase, H. (2000). Learning about interface design through peer assessment. *Assessment & Evaluation in Higher Education*, 24(4), 341–352.

Preston, G., Phillips, R., Gosper, M., McNeill, M., Woo, K., & Green, D. (2010). Web-based lecture technologies: Highlighting the changing nature of teaching and learning. *Australasian Journal of Educational Technology*, 26(6), 717–728.

Prince, M. (2004). Does active learning work? A review of the research. *Journal of Engineering Education*, 93(3), 223–231.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts by novices. *Computational Science Education*, 15(3), 203–221.

Reeves, T. C., & Reeves, P. M. (1997). Effective dimensions of interactive learning on the World Wide Web. In B. H. Khan (Ed.), *Web-Based Instruction* (pp. 59–66). Englewood Cliffs, NJ: Educational Technology Publications.

Reilly, M., Shen, H., Calder, P., & Duh, H. (2014). Towards a collaborative classroom through shared workspaces on mobile devices. In *28th International BCS Human Computer Interaction Conference on HCI 2014 – Sand, Sea and Sky – Holiday HCI (BCS-HCI '14)* (pp. 335–340). London, UK: BCS.

Reza, S., & Baig, M. (2015). A study of inverted classroom pedagogy in computer science teaching. *International Journal of Research Studies in Educational Technology*, 4(2). Retrieved from www.learntechlib.org/p/151048/

Rich, L., Perry, H., & Guzdial, M. (2004). A CS1 course designed to address interests of women. *SIGCSE Bulletin*, 36(1), 190–194.

Rizzardini, R. H., & Amado-Salvatierra, H. R. (2017). Full engagement educational framework: A practical experience for a MicroMaster. In *4th ACM Conference on Learning @ Scale (L@S '17)* (pp. 145–146). New York: ACM.

Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. In *44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)* (pp. 651–656). New York: ACM.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & De Madrid, A. P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education*, 82, 409–420.

Rudestam, K., & Schoenholtz-Read, J. (2010). The flourishing of adult online education: an overview. In K. Rudestam & J. Schoenholtz-Read (Eds.), *Handbook of Online Learning*, 2nd edn (pp. 1–28). Thousand Oaks, CA: SAGE.

Rudman, P. (2002). *Investigating Domain Information as Dynamic Support for the Learner During Spoken Conversations* (PhD thesis). University of Birmingham.

Salmon, G. (2004). *E-Tivities: The Key to Active Online Learning*, 2nd edn. Abingdon, UK: Taylor & Francis e-Library.

Sanders, K., Boustedt, J., Eckerdal, A., McCartney, R., & Zander, C. (2017). Folk pedagogy: Nobody doesn't like active learning. In *2017 ACM Conference*

on International Computing Education Research (ICER '17) (pp. 145–154). New York: ACM.

Schneider, B., & Blikstein, P. (2016). Flipping the flipped classroom: A study of the effectiveness of video lectures versus constructivist exploration using tangible user interfaces. *IEEE Transactions on Learning Technologies*, 9(1), 5–17.

Sharples, M., de Roock, R., Ferguson, R., Gaved, M., Herodotou, C., Koh, E., Kukulska-Hulme, A., Looi, C-K., McAndrew, P., Rienties, B., Weller, M. & Wong, L. H. (2016). *Innovating Pedagogy 2016: Open University Innovation Report* 5. The Open University.

Shulman, L. (2005). Pedagogies. *Liberal Education*, 91(2), 18–25.

Siemens, G. (2005). Connectivism: A learning theory for the digital age. In *International Journal of Instructional Technology and Distance Learning*, 2(1), 3–10.

Siemens, G. (2012). MOOCs are really a platform. *ELearnspace.* Retrieved from www.elearnspace.org/blog/2012/07/25/moocs-are-really-a-platform/

Simon, B., & Cutts, Q. (2012). Peer instruction: A teaching method to foster deep understanding. *Communications of the ACM*, 55(2), 27–29.

Sirkiä, T. (2016). Combining parson's problems with program visualization in CS1 context. In *16th Koli Calling International Conference on Computing Education Research (Koli Calling '16)* (pp. 155–159). New York: ACM.

Sirkiä, T., & Sorva, J. (2015). Tailoring animations of example programs. In *15th Koli Calling Conference on Computing Education Research (Koli Calling '15)* (pp. 147–151). New York: ACM.

Slavin, R. E. (1991). Synthesis of research on cooperative learning. *Educational Leadership*, 48(5), 71–82.

Slavin, R. E., & Cooper, R. (1999). Improving intergroup relations: Lessons learned from cooperative learning programs. *Journal of Social Issues*, 55, 647–663.

Smith, P. A., & Webb, G. I. (1995). Reinforcing a generic computer model for novice programmers. In *7th Australian Society for Computer in Learning in Tertiary Education Conference* (ASCILITE '95). Retrieved from www.ascilite.org/conferences/melbourne95/smtu_bak/papers/smith.pdf

Smith, J., Tessler, J., Kramer, E., & Lin, C. (2012). Using peer review to teach software testing. In *9th Annual International Conference on International Computing Education Research (ICER '12)* (pp. 93–98). New York: ACM.

Sonnentag, S. (1998). Expertise in professional software design: A process study. *Journal of Applied Psychology*, 83, 703–715.

Sorva. J. (2013). Notional machines and introductory programming education. *Transactions on Computing Education (TOCE)*, 13(2), 8.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *Transactions on Computing Education (TOCE)*, 13(4), 15.

Sorva, J., & Sirkiä, T. (2010). UUhistle: A software tool for visual program simulation. In *10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)* (pp. 49–54). New York: ACM.

Srinivasan, V., Butler-Purry, K. & Pedersen, S. (2008). Using video games to enhance learning in digital systems. In *2008 Conference on Future Play: Research, Play, Share (Future Play '08)* (pp. 196–199). New York: ACM.

Summet, J., Kumar, D., O'Hara, K., Walker, D., Ni, L., Blank, D., & Balch, T. (2009). Personalizing CS1 with robots. *ACM SIGCSE Bulletin*, 41(1), 433–437.

Swan, K., Day, S., & Bogle, L. (2016). Metaphors for learning and MOOC pedagogies. In *3rd ACM Conference on Learning @ Scale* (pp. 125–128). New York: ACM.

Tarimo, W. T., Deeb, F. A., & Hickey, T. J. (2015). A flipped classroom with and without computers. In *International Conference on Computer Supported Education* (pp. 333–347). New York: Springer International Publishing.

Titterton, N., Lewis, C. M., & Clancy, M. J. (2010). Experiences with lab-centric instruction. *Computer Science Education*, 20(2), 79–102.

Toven-Lindsey, B., Rhoads, R. A., & Lozano, J. B. (2015). Virtually unlimited classrooms: Pedagogical practices in massive open online courses. *The Internet and Higher Education*, 24, 1–12.

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, 17(4), 16.

Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, Greenfoot, and Scratch – A discussion. *ACM Transactions on Computing Education (TOCE)*, 10(4), 17.

Vassiliadis, B., Kameas, A., & Sgouropoulou, C. (2016). A closer look at MOOC's adoption from a qualitative perspective. In *20th Pan-Hellenic Conference on Informatics (PCI '16)* (p. 17). New York: ACM.

Vickers, R., Cooper, G., Field, J., Thayne, M., Adams, R., & Lochrie, M. (2014). Social media and collaborative learning: Hello Scholr. In *18th International Academic MindTrek Conference: Media Business, Management, Content & Services (AcademicMindTrek '14)* (pp. 103–109). New York: ACM.

Vozniuk, A., Holzer, A., & Gillet, D. (2014). Peer assessment based on ratings in a social media course. In *4th International Conference on Learning Analytics And Knowledge (LAK '14)* (pp. 133–137). New York: ACM.

Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press.

Wakeling, D. (2008). A robot in every classroom: Robots and functional programming across the curriculum. In *2008 International Workshop on Functional and Declarative Programming in Education (FDPE '08)* (pp. 51–60). New York: ACM.

Walker, J. D., Brooks, D. C., & Baepler, P. (2011). Pedagogy and space: Empirical research on new learning environments. *EDUCAUSE Quarterly*, 34(4). Retrieved from https://eric.ed.gov/?id=EJ958727

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. In *2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)* (pp. 39–44). New York: ACM.

Wenger, E. (1998). Communities of practice. Learning as a social system. *Systems Thinker*. 9(5). Retrieved from https://thesystemsthinker.com/communities-of-practice-learning-as-a-social-system/

Williams, L. (2000). *The Collaborative Software Process* (PhD dissertation). University of Utah.

Williams, L., & Kessler, R. (2002). *Pair Programming Illuminated*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Williams, L., McCrickard, D. S., Layman, L., & Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. In G. Melnik & M. Poppendieck (Eds.), *Agile Conference* (pp. 445–452). New York: IEEE Press.