Research Directions for Teaching Programming Online

Amber Settle[1], **Arto Vihavainen**[2], **and Craig S. Miller**[1]
[1]School of Computing, DePaul University, Chicago, Illinois, USA
[2]Department of Computer Science, University of Helsinki, Helsinki, Finland

Abstract— *Online education has a long-standing tradition in academia, and yet online programming remains a relatively undeveloped area in the computing education literature. This is in sharp contrast with in-person programming courses, which have been a favorite subject of study in computing education. Research into teaching on-site programming is so extensive that numerous practices have emerged and are typically supported by instructional theory, empirical results or both. In this article we identify some of the commonly accepted practices for teaching programming in an on-site environment and survey the work that has been done for online programming. In doing so, we contrast the rigor of on-site programming research with the relative immaturity of educational practices for online programming. We identify research questions and future directions for online programming educators, with the goal of inspiring the same high-quality work that on-site programming research has produced.*

Keywords: online programming, programming, pedagogy, tools, best practices

1. Introduction

The computing community has recently witnessed a rebirth of interest in online education. The immense popularity of Khan Academy and Code.org, and the collaboration between organizations like Coursera, edX, and Udacity and prestigious universities like MIT, Harvard, Stanford, and others have brought new energy to a long-standing educational approach. The attention from the academic community, for example in the creation of new conferences like ACM Learning@Scale, and the interest from the popular press have converged to produce diverse opportunities for reaching hundreds of thousands of students.

While the new possibilities have energized many educators in computing, lost in the enthusiasm is the acknowledgment that online education, often called distance education prior to 2000, has existed for decades. To provide just two examples of enduring online education journals, *Distance Education* was founded in 1980 and *The American Journal of Distance Education* was created in 1987. What the recent initiatives arguably contribute is large scale access to students, a new and highly interesting development in online education. There is also a much stronger focus on teaching programming in the latest round of online educational initiatives.

Despite an established body of research in online education, research in teaching programming online remains underdeveloped. Unlike traditional programming courses where scores of researchers have conducted numerous quantitative and qualitative studies of how best to teach students to program (for surveys, see [55], [49]), the research into teaching online programming rarely identifies replicable practices that are either theoretically motivated or supported by empirical study. The majority of published papers are experience reports, and such articles are difficult to adapt and generalize. This leaves educators working on teaching online programming today with little guidance in how to approach their work. However, unlike previous online education research, the stakes now are higher. Many schools are adopting programming courses into their offerings, often basing their content on existing online courses. The attention of the press may lead any mistakes to be widely reported, and momentum in reaching hundreds of thousands of interested students will not be easy to recapture.

In this article, we explore the space of traditional "on-site" or "in-person" strategies for teaching programming to understand how it is that educators have approached the problem, after which we outline research done specifically on online programming. By "online", we mean a course in which a large portion of the activities and assessments are completed by students who are not on-site. In doing so we identify a number of gaps between the work in teaching on-site and online programming and provide suggestions for how to advance research in online programming in a more methodical and rigorous direction. Doing so will allow online educators to make more significant progress than previous researchers in order to capitalize on its momentum.

2. Teaching Programming

Teaching and learning programming has been researched widely during the past few decades to the point where standard educational approaches are forming from the numerous experiments. In this section, current educational practices that are used in teaching programming are discussed, with the goal of emphasizing the increasing maturity of the area. While a comprehensive survey is not possible here, a range of

practices is presented, selected for common acceptance and diverse representation. Although the practices are presented separately, many of them have been studied and applied together to form hybrid courses (see e.g. [50]).

2.1 Changing Interaction

Classrooms have been traditionally seen as large auditoriums filled with thousands of students. The most stereotypical understanding of this situation is the classroom with a lonely teacher talking to students who struggle to follow. In response, computing education researchers have identified numerous ways in which students can be more fully engaged in learning to program, either by increasing student to student interaction, student to instructor interaction, or student to content interaction [45], [1]. Many of the approaches use active learning, where students not only do, but think about what they are doing [7].

Peer instruction [42] was originally created to solve the passive classroom-issue in physics education [19]. In order to have better participation from students, they prepare for lectures beforehand by reading provided material. As students have already read the material, the lectures are devoted to the teacher discussing the content, and engaging the students with interactive questions from which both the teacher and students benefit. Although some studies suggest that quizzes in lectures are not efficient [43], the questions in peer instruction are done in multiple iterations where the students think both alone and collaboratively in groups. Peer instruction helps instructors adapt their class to address student misconceptions, increase student retention, and improve learning outcomes over traditional lectures [19], [56].

Another approach to tackling the passive classroom is to move to hands-on programming assignments as soon as possible [67]. Students benefit immensely if instruction is moved from large classes to smaller classrooms with integrated labs and hands-on activities [69], [10], [67]. Methodologies such as lab-centric instruction [15], [62], [63], [39] and studio-based instruction [13] provide ways in which this can be done; typically students work on tasks under guidance from instructors and peers that provide support for the students if needed.

2.2 Collaboration and Peer Support

In order to have multiple directions for input, peer reviews can be utilized given that the guidelines emphasize the learning objectives [4]. Peer evaluation is often used in studio-based classroom settings [31] and, in one context, the quality of students' reviews have been found comparable to those provided by paid teaching assistants in computer science [29].

Work in laboratories can be done in pairs or in small groups [69] and organized to emphasize good programming practices such as the ones outlined by extreme programming [57]. The most typical extreme programming practice taken into introductory programming laboratories is pair programming, where pairs of students work together to create programs. As two students work on one assignment, the work is constantly being reviewed, and possible challenges can be discussed directly with the peer. In various studies, pair programming has been found to improve students' performance and retention [47], [11].

Additional collaboration and peer support can be facilitated by recruiting students as teaching assistants and mentors. In peer-led team learning, students work in groups that are led by e.g. peers or undergraduate teaching assistants, who are typically trained and supervised by faculty [30], [32] and are also themselves learning during the process. Peer-led team learning reworks student interactions significantly, although it is evident that recruiting the correct people and providing them proper guidance is of the utmost importance. Peer-led team learning has been reported to increase student retention [58] as well as increase the amount of students that choose CS as a major [46]. In addition, the use of undergraduate teaching assistants has been reported to create a more enjoyable context [20] as well as to help students improve their teaching and mentoring skills [68].

2.3 Modeling Problem Solving

When students are learning to program, they benefit immensely if they have a model on how a programming problem should be solved. Some modeling can done by process recordings [6] and by task design (see e.g. [65]), while for some, the use of a virtual classroom can be beneficial [8]. One suggestion is to utilize case studies [40], which consist of a problem and a narrative description on how solutions for the problems have been reached; case studies can be also seen as worked examples [14].

Lectures can be also used for worked examples [14], during which the teacher discusses a problem and works through it in a step-by-step fashion. Extensive empirical study has shown that novices greatly benefit from seeing worked examples in the early stages of learning [37], [44]. Worked examples are one approach for providing students a mental model, an explanation of the thought process that is needed to solve a problem. The need for a mental model is emphasized in teaching approaches that rely on cognitive apprenticeship, which is a theory on the process of how a more experienced person can teach a skill to a novice [16]. Cognitive apprenticeship has been adapted and discussed in the programming instruction-context (see e.g. [2], [14], [67]).

2.4 Engaging Content and Contextualization

Activity in classrooms can also be increased via the use of engaging content. As an example, in media computation [27], [24], students are introduced to computing using an interesting context; multimedia. The lectures often evolve around live-coding [3] examples, where the lecturer creates representations such as music or graphics. Media computing

has been reported to increase student retention across various tertiary education institutions [28].

The way the material and assignments are designed play a large role since they form the main media with which students spend their effective learning time. As no material fits all contexts, it is important that it is contextualized so that the students can relate to it (see e.g. [64]) and that it supports the incremental steps in learning to program.

2.5 Adaptive Teaching

Adjusting the difficulty level of a course to match the students' ability level is important. Many authors suggest taking an incremental approach to building the material, which allows students to slowly build up their knowledge and skills (see e.g. [67]). This can be reflected both in the tasks that the students do, where assignments can be split into smaller parts that form subgoals, and also in the material, where one can e.g. emulate mini-languages by first only focusing on a small subset of a language [12].

Mini-languages typically have a small syntax and simple semantics, which makes starting programming easier. Mini-languages have been combined with visual programming environments [17]; the use of environments such as Scratch for teaching at-risk students has been noted to increase success in introductory programming courses [53].

3. Teaching Programming Online

As we have already noted, distance-education has a long history of study. At the same time, as our review illustrated, the study of teaching programming on-site has produced a variety of practices that are well supported both theoretically and empirically. In contrast, the study of online programming courses is still underdeveloped in the literature. As an example, performing a query[1] for articles on the ACM Digital Library related to teaching programming online using keywords "(programming OR cs1) AND (online OR distance) AND (teaching OR education)" returns 159 results, while the query "(programming OR cs1) AND (teaching OR education)" returns 1850 results.

Only a small part of the 159 articles contain somewhat relevant results, while still outlining mostly experience reports. Only a handful sought to identify underlying causes. Examples of the experience reports contain a description of how a consensus on online course structure was formed [52], what observations educators received from using specific platforms [51] and languages [60], and how the capabilities of web (i.e. no need for downloading software, dynamic sites) can be leveraged to bring more value to students [72] and to increase collaboration [34]. Some discussion on how to design and implement materials to support students learning to program exists as well, whether in an online environment or as a supplement to a more traditional course [22], [33].

[1]The queries reported in this section have been performed in April 2014

Various studies have considered differences between students in local and online classrooms. While learning outcomes are often highly related to student motivation and reasons for participation as well as course outcomes, a meta-analysis on online learning that also included non-programming-related articles suggests that instructor-directed and collaborative instruction provides better results than independent study, and that the practice of providing quizzes does not seem to be more effective than assigning homework [43]. One should also remember that online participants are often adult students who cannot attend the traditional classroom environment due to work or family-related constraints [5], [38].

When switching to an online environment, researchers have also considered the broader implications to pedagogies for both programming and in general [8]. One author suggests that an online environment could enhance students' skills such as collaborative problem solving and the capacity to integrate into new communities [8], and the same author considered the challenges of doing group work in synchronous online programming classes, developing a set of recommendations for instructors wishing to include group work in virtual classrooms [9].

3.1 MOOCs in Introductory Programming

MOOCs in programming are discussed as a separate subsection due to their inherent issue in scale and population. MOOCs in general are offered to anyone willing to participate, increasing the amount of students, and hence, creating additional challenges on making support resources available to participants. When comparing published research on MOOCs in introductory programming to the studies on teaching programming online, the amount of MOOC articles is – not surprisingly – lower. Searching for keywords "MOOC" and "programming" in ACM Digital Library produced 58 hits. Out of those articles, 14 include keywords "MOOC", "programming", "CS1", and only a few actually discuss MOOCs in introductory programming. Some of the articles discuss overall experiences in creating more advanced MOOCs (e.g. courses on software engineering) and mention introductory programming (see e.g. [18]), while some discuss possible challenges related to offering MOOCs to specific populations such as K-12 students (see e.g. [35]).

Only a few articles that describe the creation and evaluation of a MOOC in introductory programming exist. One article describes support mechanisms and activities used in a MOOC in programming that was directed to high-school students [65], while a follow-up article discussed how students recruited via the MOOC had fared in their CS studies when compared to other students [66]. While the former studies are somewhat constrained due to the low amount of participants, more recent studies have given insight on larger courses. Recently, a course that focuses on emphasizing human-human interaction in a programming MOOC [71] was

published; the team discusses their platform in a separate article [59]. Another laudable MOOC-effort is the *Functional Programming Principles in Scala*-course, which was offered at Udacity [48].

To date, only one article has evaluated several MOOCs for teaching programming; the result from the evaluation was that MOOCs did not leverage the years of research into teaching programming [4].

3.2 Tools

Nearly any search for information about online programming will produce results about tools. Unfortunately, many of the resulting papers do not address online programming courses, instead focusing on online tools for traditional programming courses. This is not simply a side effect of poor searching techniques, as many of the articles contain the words "online" and/or "programming" in the title.

As an example, consider the top fifteen articles discussing tools returned in a search of the ACM Digital Library performed using the keywords "online programming." Of these, eleven of them discussed systems designed to be used exclusively in on-site programming courses. The remaining four were only tangentially related to online programming courses. One was a proposal for a prototype-only massive multiplayer online role playing game, although the paper made no indication that the game had been fully implemented and deployed and did not suggest that the programming courses that used it would be for anything other than students physically present in a classroom[41]. Two described tools for "blended courses" in which the tool was the only online component[25], [54]. The fourth analyzed how various online assessment systems could be used for a MOOC and argued that the authors' tool was superior in many ways to other assessment systems, although the tool had not been tested in a MOOC[61]. A common direction for recent work on tools is to produce online IDEs, which was also represented in the results of the query[25].

While the creation of tools that enable better student learning is important for both on-site and online programming situations, it appears that the majority of tools with the label online are in fact tools for teaching on-site programming. For more information on tools used in introductory programming, see [49].

3.3 Suggestions from the Literature

The reviewed literature for online programming in many cases contained suggestions and guidelines for creating courses that were not supported by rigorous study. This is a common situation for experience reports, which serve as guide into new directions for educators. However, when considering the work on online programming education the suggestions one finds seem obvious. For example, the recommendations in the online programming literature include:

- Learn to utilize the capabilities of web; make the most of videos, audio and interactive content to engage multiple sensors; identify appropriate tools [70], [72], [36], [60], [21]
- Foster regular interaction and collaboration among the students to help them stay engaged; if needed, use a LMS or some other tool to provide facilities for discussion [9], [70], [51], [21]
- Ask and provide feedback; feedback can be used to both support students in their learning and to improve the course content [21]; respond to student inquiries in a timely fashion [60]
- Design the learning tasks to be informative and make the content incremental; if possible, use a tool that provides runtime syntax and error checking [72], [65], [38], [9]

These recommendations, which are common sense for experienced educators, apply equally well to both online and on-site courses. More importantly, they provide a strong contrast with the recommendations for teaching on-site programming, which are based on theories about learning and/or have been empirically tested for validity. This is an indicator of the relative immaturity of the online programming literature.

4. Discussion of Unresolved issues

Experience reports often serve a good purpose, for example by demonstrating that an approach or idea has merit for further consideration. The next step to take is a more rigorous investigation of pedagogies, tools, and learning environments as well as learner perspectives, which seek to provide answers and information for adoption. In this paper, we turned to the extensive research on teaching programming in traditional contexts for direction. Its offering of teaching practices, derived from instructional theory and based on empirical study, provides useful goals for the less developed study of teaching programming online.

Teaching programming online is an area where experience reports still heavily predominate the literature, indicating that the research still remains in the early stages. Part of the reason may be that many educators have their first teaching experiences in a face-to-face environment. This makes it natural to replicate that experience when creating online courses, which requires using tools to produce the types of interaction seen in the on-site classroom. While approaching teaching programming online as a translation process from existing face-to-face experiences is natural, it may be also limited by personal experiences and things that we do not know [26]. Moving forward from replicating in-person experiences might help online programming classes to take advantage of the unique strengths of the online environment.

The evolution of online programming as a research area would involve investigating many of the questions that have been studied for on-site programming courses, including:

- How can interaction, whether student-student, student-instructor, or student-content, be improved?
- How can tools be used to improve interaction without unduly increasing cognitive load? Comparative studies are especially useful for this question.
- How does learning programming in an online environment affect student course outcomes in later courses and projects?
- How effective are various pedagogies, tools, and techniques? Here empirical studies that can be replicated are a necessity.
- What approaches can be shown to work in comparative studies, whether across student populations, institutions, or cultures?

Note that these points call for studies that evaluate the effectiveness of practices and their online counterparts. While previous studies have evaluated online learning at the course-level (see e.g. [38], [23]), practice-level studies will not only provide explanations for why one course delivery is better than another, it will provide directions for further improvement for diverse modes of delivery.

While investigating existing research questions is useful for the online context, there are some questions that are unique to online programming courses such as:

- What is the relationship between the scale of an online programming class and the amount and type of interaction found in that class?
- How do technological advances change the utility of the results in papers related to teaching programming online? What could be done to better withstand the influence of time and technological change?
- What impact does the audience for an online programming class have on the approaches used? Heterogeneous populations, particularly for massive online classes, complicate research studies.

Advancing the body of knowledge in this area requires both consideration of existing work as well as creative out-of-the-box thinking for the design of online programming classes. There is nothing wrong with finding motivation from the latest tools and technology, provided that the resulting course activities produce the desired learning objectives. But a first step in reaching a learning objective is to define one, and this is an area where researchers in online programming have much left to do.

References

[1] T. Anderson. Getting the mix right again: An updated and theoretical rationale for interaction. *The International Review of Research in Open and Distance Learning*, 4(2), 2003.

[2] O. Astrachan and D. Reed. Aaa and cs 1: the applied apprenticeship approach to cs 1. In *ACM SIGCSE Bulletin*, volume 27, pages 1–5. ACM, 1995.

[3] L. J. Barker, K. Garvin-Doxas, and E. Roberts. What can computer science learn from a fine arts approach to teaching? *SIGCSE Bull.*, 37(1):421–425, Feb. 2005.

[4] M. Ben-Ari. MOOCs on introductory programming: A travelogue. *ACM Inroads*, 4(2):58–61, June 2013.

[5] K. Benda, A. Bruckman, and M. Guzdial. When life and learning do not fit: Challenges of workload and communication in introductory computer science online. *Trans. Comput. Educ.*, 12(4):15:1–15:38, Nov. 2012.

[6] J. Bennedsen and M. E. Caspersen. Exposing the programming process. In *Reflections on the Teaching of Programming*, pages 6–16. Springer, 2008.

[7] C. C. Bonwell and J. A. Eison. *Active learning: Creating excitement in the classroom*. School of Education and Human Development, George Washington University Washington, DC, 1991.

[8] M. Bower. Virtual classroom pedagogy. *SIGCSE Bull.*, 38(1):148–152, Mar. 2006.

[9] M. Bower. Groupwork activities in synchronous online classroom spaces. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 91–95, New York, NY, USA, 2007. ACM.

[10] K. E. Boyer, R. S. Dwight, C. S. Miller, C. D. Raubenheimer, M. F. Stallmann, and M. A. Vouk. A case for smaller class size with integrated lab for introductory computer science. *SIGCSE Bull.*, 39(1):341–345, Mar. 2007.

[11] G. Braught, T. Wahls, and L. M. Eby. The case for pair programming in the computer science classroom. *Trans. Comput. Educ.*, 11(1):2:1–2:21, Feb. 2011.

[12] P. Brusilovsky, E. Calabrese, J. Hvorecky, A. Kouchnirenko, and P. Miller. Mini-languages: a way to learn programming principles. *Education and Information Technologies*, 2(1):65–83, 1997.

[13] A. S. Carter and C. D. Hundhausen. A review of studio-based learning in computer science. *J. Comput. Sci. Coll.*, 27(1):105–111, Oct. 2011.

[14] M. E. Caspersen and J. Bennedsen. Instructional design of a programming course: a learning theoretic approach. In *Proceedings of the third intl workshop on Computing education research*, pages 111–122. ACM, 2007.

[15] M. Clancy, N. Titterton, C. Ryan, J. Slotta, and M. Linn. New roles for students, instructors, and computers in a lab-based introductory programming course. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 132–136, New York, NY, USA, 2003. ACM.

[16] A. Collins, J. S. Brown, and A. Holum. Cognitive apprenticeship: making thinking visible. *American Educator*, 6:38–46, 1991.

[17] S. Cooper, W. Dann, and R. Pausch. Using animated 3d graphics to prepare novices for cs1. *Computer Science Education*, 13(1):3–30, 2003.

[18] S. Cooper and M. Sahami. Reflections on stanford's moocs. *Commun. ACM*, 56(2):28–30, Feb. 2013.

[19] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69:970, 2001.

[20] P. E. Dickson. Using undergraduate teaching assistants in a small college environment. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 75–80. ACM, 2011.

[21] E. M. El-Sheikh. Techniques for engaging students in an online computer programming course. *Journal of Systemics, Cybernetics & Informatics*, 7(1), 2009.

[22] A. Ellis, D. Hagan, J. Sheard, J. Lowder, W. Doube, A. Carbone, J. Robinson, and S. Tucker. A collaborative strategy for developing shared java teaching resources to support first year programming. In *Proceedings of the 4th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '99, pages 84–87, New York, NY, USA, 1999. ACM.

[23] W. Farag. Comparing achievement of intended learning outcomes in online programming classes with blended offerings. In *Proceedings of the 13th Annual Conf. on Information Technology Education*, SIGITE '12, pages 25–30, New York, NY, USA, 2012. ACM.

[24] A. Forte and M. Guzdial. Computers for communication, not calculation: Media as a motivation and context for learning. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii Intl. Conf. on*, pages 10–pp. IEEE, 2004.

[25] S. Friese. Measuring of and reacting to learners' progress in logic programming courses. In *Proceedings of the Fifteenth Annual Confer-*

ence on Innovation and Technology in Computer Science Education, ITiCSE '10, pages 152–154, New York, NY, USA, 2010. ACM.

[26] J. Gal-Ezer and D. Harel. What (else) should cs educators know? *Commun. ACM*, 41(9):77–84, Sept. 1998.

[27] M. Guzdial. A media computation course for non-majors. *SIGCSE Bull.*, 35(3):104–108, June 2003.

[28] M. Guzdial. Exploring hypotheses about media computation. In *Proceedings of the Ninth Annual Intl. ACM Conf. on Intl. Computing Education Research*, ICER '13, pages 19–26, New York, NY, USA, 2013. ACM.

[29] J. Hamer, H. C. Purchase, P. Denny, and A. Luxton-Reilly. Quality of peer assessment in cs1. In *Proceedings of the Fifth Intl. Workshop on Computing Education Research Workshop*, ICER '09, pages 27–36, New York, NY, USA, 2009. ACM.

[30] S. Horwitz, S. H. Rodger, M. Biggers, D. Binkley, C. K. Frantz, D. Gundermann, S. Hambrusch, S. Huss-Lederman, E. Munson, B. Ryder, et al. Using peer-led team learning to increase participation and success of under-represented groups in introductory computer science. In *ACM SIGCSE Bull.*, volume 41, pages 163–167. ACM, 2009.

[31] C. Hundhausen, A. Agrawal, D. Fairbrother, and M. Trevisan. Integrating pedagogical code reviews into a cs 1 course: An empirical study. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 291–295, New York, NY, USA, 2009. ACM.

[32] S. Huss-Lederman, D. Chinn, and J. Skrentny. Serious fun: peer-led team learning in cs. In *ACM SIGCSE Bulletin*, volume 40, pages 330–331. ACM, 2008.

[33] L. J. White, J. W. Coffey, and E. M. El-Sheikh. Exploring technologies, materials, and methods for an online foundational programming course. *Informatics in Education*, (Vol 7:2):259–276, 2008.

[34] J. Jenkins, E. Brannock, T. Cooper, S. Dekhane, M. Hall, and M. Nguyen. Perspectives on active learning and collaboration: Javawide in the classroom. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 185–190, New York, NY, USA, 2012. ACM.

[35] L. C. Kaczmarczyk. We need to talk. *ACM Inroads*, 5(1):30–31, Mar. 2014.

[36] R. Karsten, S. Kaparthi, and R. M. Roth. Teaching programming via the web: A time-tested methodology. *College Teaching Methods & Styles Journal (CTMS)*, 1(3):73–82, 2005.

[37] P. A. Kirschner, J. Sweller, and R. E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86, 2006.

[38] J. Kleinman and E. B. Entin. Comparison of in-class and distance-learning students' performance and attitudes in an introductory computer science course. *J. Comput. Sci. Coll.*, 17(6):206–219, May 2002.

[39] C. M. Lewis, N. Titterton, and M. Clancy. Developing students' self-assessment skills using lab-centric instruction. *J. Comput. Sci. Coll.*, 26(4):173–180, Apr. 2011.

[40] M. C. Linn and M. J. Clancy. The case for case studies of programming problems. *Commun. ACM*, 35(3):121–132, Mar. 1992.

[41] C. Malliarakis, M. Satratzemi, and S. Xinogalos. Towards a new massive multiplayer online role playing game for introductory programming. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 156–163, New York, NY, USA, 2013. ACM.

[42] E. Mazur and R. C. Hilborn. Peer instruction: A user's manual. *Physics Today*, 50(4):68–69, 1997.

[43] B. Means, Y. Toyama, R. Murphy, M. Bakia, and K. Jones. Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies. 2010.

[44] C. S. Miller and A. Settle. When practice doesn't make perfect: Effects of task goals on learning computing concepts. *ACM Transactions on Computing Education (TOCE)*, 11(4):22, 2011.

[45] M. G. Moore. Editorial: Three types of interaction. *American Journal of Distance Education*, 3(2):1–7, 1989.

[46] C. Murphy, R. Powell, K. Parton, and A. Cannon. Lessons learned from a pltl-cs program. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 207–212, New York, NY, USA, 2011. ACM.

[47] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik. Improving the cs1 experience with pair programming. In *ACM SIGCSE Bulletin*, volume 35, pages 359–362. ACM, 2003.

[48] M. Odersky, L. Rytz, H. Miller, and P. Haller. Functional programming for all! scaling a mooc for students and professionals alike. In *36th International Conference on Software Engineering (ICSE" 14) SEET Track*, number EPFL-CONF-190022, 2014.

[49] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '07, pages 204–223, New York, NY, USA, 2007. ACM.

[50] L. Porter and B. Simon. Retaining nearly one-third more majors with a trio of instructional best practices in cs1. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 165–170, New York, NY, USA, 2013. ACM.

[51] T. Reeves, P. Baxter, and C. Jordan. Teaching computing courses - computer literacy, business microcomputer applications, and introduction to programming online utilizing webct. *J. Comput. Sci. Coll.*, 18(1):290–300, Oct. 2002.

[52] K. Renaud, J. Barrow, and P. le Roux. Teaching programming from a distance: Problems and a proposed solution. *SIGCSE Bull.*, 33(4):39–42, Dec. 2001.

[53] M. Rizvi and T. Humphries. A scratch-based cs0 course for at-risk computer science majors. In *Proceedings of the 2012 IEEE Frontiers in Education Conference (FIE)*, FIE '12, pages 1–5, Washington, DC, USA, 2012. IEEE Computer Society.

[54] G. H. B. Roberts and J. L. M. Verbyla. An online programming assessment tool. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20*, ACE '03, pages 69–75, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[55] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[56] B. Simon, M. Kohanfars, J. Lee, K. Tamayo, and Q. Cutts. Experience report: Peer instruction in introductory computing. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 341–345. ACM, 2010.

[57] S. Smith and S. Stoecklin. What we can learn from extreme programming. *J. Comput. Sci. Coll.*, 17(2):144–151, Dec. 2001.

[58] C. Stewart-Gardiner. Improving the student success and retention of under achiever students in introductory computer science. *J. Comput. Sci. Coll.*, 26(6):16–22, June 2011.

[59] T. Tang, S. Rixner, and J. Warren. An environment for learning interactive programming. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 671–676, New York, NY, USA, 2014. ACM.

[60] R. Thomas. Experiences teaching c++ programming online. In *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, CCSC '00, pages 211–219, USA, 2000. Consortium for Computing Sciences in Colleges.

[61] N. Tillmann, J. De Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1117–1126, Piscataway, NJ, USA, 2013. IEEE Press.

[62] N. Titterton and M. J. Clancy. Adding some lab time is good, adding more must be better: the benefits and barriers to lab-centric courses. In *FECS*, pages 363–367, 2007.

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

[64] M. Vesisenaho, M. Duveskog, E. Laisser, and E. Sutinen. Designing a contextualized programming course in a tanzanian university. In *Frontiers in Education Conference, 36th Annual*, pages 1–6. IEEE, 2006.

[65] A. Vihavainen, M. Luukkainen, and J. Kurhila. Multi-faceted support for mooc in programming. In *Proceedings of the 13th annual conference on Information technology education*, pages 171–176. ACM, 2012.

[66] A. Vihavainen, M. Luukkainen, and J. Kurhila. Mooc as semester-long entrance exam. In *Proceedings of the 14th Annual ACM SIGITE*

Conference on Information Technology Education, SIGITE '13, pages 177–182, New York, NY, USA, 2013. ACM.

[67] A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 93–98. ACM, 2011.

[68] A. Vihavainen, T. Vikberg, M. Luukkainen, and J. Kurhila. Massive increase in eager tas: Experiences from extreme apprenticeship-based cs1. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 123–128, New York, NY, USA, 2013. ACM.

[69] H. M. Walker. Collaborative learning: A case study for cs1 at grinnell college and austin. *SIGCSE Bull.*, 29(1):209–213, Mar. 1997.

[70] W. Wang. Teaching programming online. In *International Conference on the Future of Education*, 2011.

[71] J. Warren, S. Rixner, J. Greiner, and S. Wong. Facilitating human interaction in an online programming course. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 665–670, New York, NY, USA, 2014. ACM.

[72] J. L. Zachary and P. A. Jensen. Exploiting value-added content in an online course: Introducing programming concepts via html and javascript. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 396–400, New York, NY, USA, 2003. ACM.