Tools and Techniques for Teaching Computer Programming: A Review

Article *in* Journal of Educational Technology Systems · May 2020

DOI: 10.1177/0047239520926971

Kanika Kanika
Netaji Subhas Institute of Technology
10 PUBLICATIONS **42** CITATIONS

Article

Journal of Educational Technology Systems
0(0) 1–29
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0047239520926971
journals.sagepub.com/home/ets
SAGE

Tools and Techniques for Teaching Computer Programming: A Review

Kanika[1], Shampa Chakraverty[1], and Pinaki Chakraborty[1]

Abstract
Courses on computer programming are included in the curricula of almost all engineering disciplines. We surveyed the research literature and identified the techniques that are commonly used by instructors for teaching these courses. We observed that visual programming and game-based learning can enhance computational thinking and problem-solving skills in students and may be used to introduce them to programming. Robot programming may be used to attract students to programming, but the success of this technique is subjected to the availability of robots. Pair and collaborative programming allows students to learn from one another and write efficient programs. Assessment systems help instructors in evaluating programs written by students and provide them with timely feedback. Furthermore, an analysis of citations showed that Scratch is the most researched tool for teaching programming. We discuss how these techniques may be used to teach introductory courses, advanced courses, and massive open online courses on programming.

Keywords
computer science education, visual programming, game-based learning, pair programming, collaborative learning, robot programming, assessment system

[1]Department of Computer Science and Engineering, Netaji Subhas University of Technology, New Delhi, India

Corresponding Author:
Pinaki Chakraborty, Netaji Subhas University of Technology, Sector 3, Dwarka, New Delhi, 110078, India.
Email: pinaki_chakraborty_163@yahoo.com

The concept of programming is at the core of computer science. Computer engineers write programs that are executed by others on their computers to perform various computational tasks. Because computers and computer-based methodologies are now used in all branches of engineering, undergraduate students of all engineering disciplines are taught programming in most universities (Peteranetz et al., 2018). Engineers should be knowledgeable about the logic and techniques used to write computer programs. An undergraduate program in computer science invariably includes several programming-related courses of increasing difficulty level (Liu, 2014). Alternatively, at least one introductory course on computer programming is included in the undergraduate curriculum of other engineering disciplines (Wang et al., 2017).

The introductory course on programming is typically taught in the early part of an undergraduate engineering program and influences several later courses. The course is important because of the following reasons:

- It exposes students to the terminology and the fundamental concepts of programming.
- It enhances computational thinking and problem-solving abilities in students.
- It teaches students to design, implement, test, and debug a program.
- It enlightens students about the good practices in programming.

The course is taught using a particular programming language. Students learn about data types, variable declaration, operators and expressions, input and output statements, conditional and iterative statements, arrays and strings, pointers, user-defined data types, functions and recursion, and file handling (Yuan et al., 2015). Historically, many programming languages have been used to teach the course. However, C++, Java, and Python are used in most universities today (Kunkle & Allen, 2016). Instructors typically emphasize equally on the fundamental concepts of programming and the syntax of the programming language while teaching the course. Alternatively, advanced courses on computer programming discuss concepts such as object-oriented programming, design patterns, multithreaded programming, event handling, and programming a windowing system. Courses on data structures and algorithms are also included in the curriculum.

The introductory and advanced courses on programming are being taught since the mid-20th century. Over the years, computer hardware and programming languages have evolved, and the complexity of the tasks being solved using computers has increased. Accordingly, the techniques used for writing programs have also improved gradually. Teaching a course on programming is a complex task and has its own challenges (Koulouri et al., 2014). The instructor has to teach students to analyze a problem logically and express their thought process in a programming language. For many years, educationists have been researching specialized tools and techniques for teaching courses on programming because of their unique requirements and their strong influence on other courses. Research on tools and techniques to teach programming has become more systematic and is being reported in the literature with more emphasis, since the early 1990s. In this article, we review the prominent techniques used for teaching programming and discuss how they should be used. The purpose of this review is to inform instructors about, and encourage them to use, the tools and techniques available for teaching courses on programming.

Review and Classification Strategy

We surveyed the literature for papers discussing novel approaches for teaching introductory as well as advanced courses on programming. Papers introducing novel tools for teaching programming and papers reporting the effect of such tools on the performance of students were included in the review. Papers discussing general issues related to teaching computer programming to students and papers related to courses on data structures and algorithms were excluded from the review. Most of the papers we thus included in the review were published after 1990.

We clustered the papers according to the approaches used in them to teach programming. Five such clusters of papers were formed representing five techniques that are commonly used for teaching programming as follows:

- *Visual programming*: A program is represented by a flowchart-like diagram in a visual programming language. Students learn programming by interactively drawing such diagrams. Specialized integrated development environments (IDEs) are required to use this technique.
- *Game-based learning*: Students learn programming by writing programs to play visual games in suitably designed environments.
- *Pair and collaborative programming*: Students write programs in pairs or small groups. They get quick feedback and learn from one another. Collaborative IDEs are sometimes used to support this technique.
- *Robot programming*: Students write programs to control robots and make them perform simple tasks in the real world. The robots are specially built for educational purposes and are inexpensive.
- *Assessment systems*: Programs written by students are checked and graded by suitably designed software tools. Some assessment systems also provide feedback in natural language.

Visual Programming

Computers with sophisticated graphical user interface became widespread in the 1980s. This prompted instructors teaching courses on programming to develop visual tools for teaching their courses. Such a visual tool provides a learner with

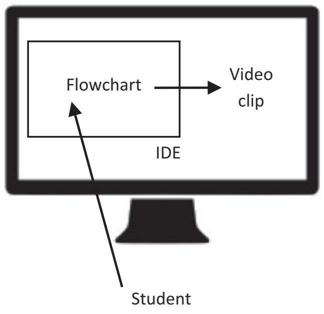
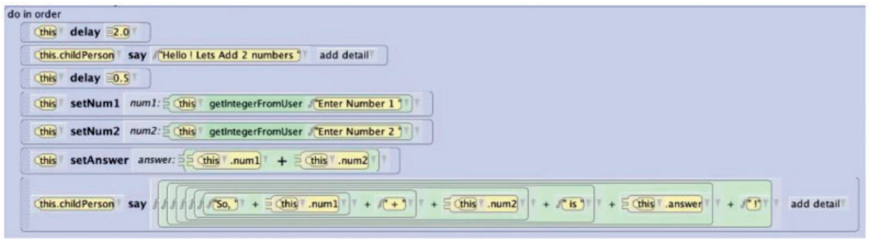


Figure 1. Use of Visual Language to Teach Programming.
IDE = integrated development environment.

a list of predefined programming constructs such as conditional statements, iterative statements, and subroutine calls. The programming constructs are represented in the tool by suitable icons. A learner *writes* a program by dragging and dropping necessary programming constructs on a canvas. The program is represented by a diagram that looks like a flowchart but is more detailed and unambiguous (Figure 1). Such a tool also includes an interpreter to execute the program. This visual approach is typically used to teach introductory courses on programming to schoolchildren, undergraduate students, and students with non-computer science background. Visual tools make it easier for learners to understand the basic concepts of programming. Visual tools also make the courses more interactive. As a result, a number of such tools have been developed.

In an early study, Glinert and Tanimoto (1984) developed a tool named Pict that provided programmers with a list of predefined functions. The predefined functions were represented by icons. A programmer could write a program by selecting the necessary functions and ordering them appropriately. The tool was suitable for introducing learners to programming. However, it did not support textual programming and could be used to write small programs only (Deek & McHugh, 1998).

More such visual tools were developed in the 1990s. Alice was developed in this decade by Pausch et al. (1995) and is undoubtedly one of the most sophisticated tools for teaching programming. Alice allows a learner to write a program interactively by manipulating visual objects in a rich three-dimensional scene (Figure 2). Running a program written in Alice looks like an animated film or an interactive visual game. Alice helps novice learner to gain a better understanding of the basic concepts of programming (Biju, 2013). Alice has been used extensively, at various levels, around the world for decades (Cooper et al., 2000). Alice also allows writing code in a textual format to manipulate the visual objects. This combination of visual and textual programming makes Alice a



```
do in order
  this delay 2.0
  this.childPerson say "Hello ! Lets Add 2 numbers " add detail
  this delay 0.5
  this setNum1 num1: this getIntegerFromUser "Enter Number 1 "
  this setNum2 num2: this getIntegerFromUser "Enter Number 2 "
  this setAnswer answer: this .num1 + this .num2
  this.childPerson say "So, " + this .num1 + " + " + this .num2 + " is " + this .answer + " !" add detail
```

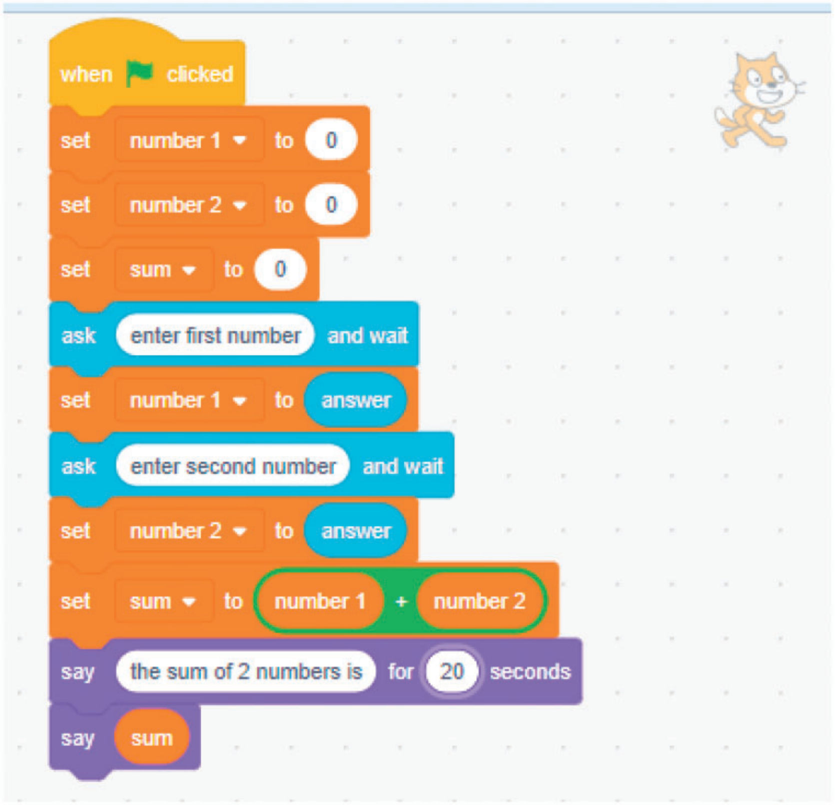
Figure 2. A Program to Calculate the Sum of Two Numbers in Alice.

powerful instructional aid. Alice has also been used to teach object-oriented programming (Cohen, 2013) and advanced topics in programming (Wolz et al., 2009).

Several other tools to teach programming visually were developed in the 1990s. Bell and Lewis (1993) developed a tool that allowed learners to write a program diagrammatically. Freund and Roberts (1996) observed that students learning programming are often perturbed by complicated and unfriendly programming environments. Consequently, they used an interactive C interpreter to teach programming. The interpreter could be personalized according to the preferences of individual learners. DigGiano (1996) developed a tool that allowed writing programs as a combination of diagrams and text. Students typically started by representing their programs diagrammatically and gradually moved toward writing programs in a textual format. Miyadera et al. (1996) developed a similar tool to teach fundamentals of programming. They emphasized on the modularity of programs. Bagert and Calloni (1999) developed a sophisticated tool that allowed learners to diagrammatically represent a program, and it automatically translated the program into a textual form. The tool supported procedural and objected-oriented paradigms of programming.

The 2000s saw the development of a number of visual tools and their widespread use in teaching courses on programming. Scratch was developed by Maloney et al. (2004) and is the best known tool developed in this decade. Scratch can be used to introduce children to programming (Resnick et al., 2009). A Scratch program is a flowchart-like diagram which when executed typically looks like a video clip. Scratch allows a learner to write a program interactively by dragging and dropping blocks representing different programming constructs on a canvas (Figure 3). The properties of the blocks can be then modified as per the requirements of the program. Scratch has been used successfully for teaching programming to 8- to 16-year-old children (Dann & Pausch, 2012; Malan & Leitner, 2007; Maloney et al., 2010).

Several other researchers developed visual tools of different types for teaching programming in the 2000s. Davenport (2000) used an integrated visual programming environment, named DELPHI, for teaching introductory



```
when [flag] clicked
set number 1 to 0
set number 2 to 0
set sum to 0
ask enter first number and wait
set number 1 to answer
ask enter second number and wait
set number 2 to answer
set sum to number 1 + number 2
say the sum of 2 numbers is for 20 seconds
say sum
```

Figure 3. A Program to Calculate the Sum of Two Numbers in Scratch.

programming courses. Alternatively, Mulholland and Watt (1998) developed a visual tool to teach programming to psychology students and other learners from non-computer science background. Hartmann et al. (2001) developed another visual tool to teach programming to learners from non-computer science background. Learners needed to program a virtual toy in a virtual environment. Slator et al. (2004) also tried to teach programming to students by making them program in a virtual environment. Burnett et al. (2001) developed a visual programming language to teach advanced programmers how to write code fragments to process data in a spreadsheet. The language focused on efficient flow of control, data abstraction, and graphical output. Carr et al. (2003) developed a thread library for C++. The library contained primitives that allow visualization of the internal working of a program when it is executed. Kelleher and Pausch (2005) proposed teaching programming using a simple and interactive visual language. Cook (2008) followed a similar approach and found it to be

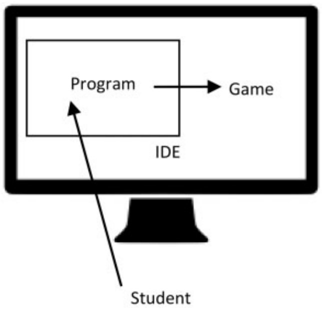


Figure 4. Use of Game-Based Learning Approach to Teach Programming.
IDE = integrated development environment.

useful for teaching introductory concepts of programming to students. Abenza et al. (2008) developed a graphical IDE that can facilitate teaching programming in Java.

Educationists continued to use visual tools to teach programming in this decade. Kordaki (2010) developed a tool to teach programming using geometric objects. Yadin (2011) improved the methodology used by Kelleher and Pausch (2005) and taught programming using simple visual languages. Tools and techniques for teaching programming visually have become sophisticated over the years. Mascarell (2011), Suo (2012), and Liang et al. (2013) have recently used visual tools to teach advanced concepts of programming.

Game-Based Learning

Since the mid-1990s, educationists have been using game-based approaches to teach programming (Figure 4). Two variants of game-based learning are used for teaching programming as follows:

- Instructors recommend students to enhance their programming skills by writing programs to play games in a suitably designed environment (Corral et al., 2014; Malliarakis et al., 2014).
- Instructors ask students to develop simple games using graphics libraries of conventional programming languages (Dolgopolovas et al., 2018; Martins et al., 2018).

These approaches are known to attract and motivate students to learn programming.

In an early study, Kato and Ide (1995) taught programming using a software tool to simulate wrestling matches in which students had to write programs to control a wrestler. Pane (2002) taught programming to schoolchildren by making them develop simple games. Similar approaches were used by several contemporary researchers. Bierre and Phelps (2004) taught programming by making students write programs in Java to display and manipulate three-dimensional objects. Leutenegger and Edgington (2007) taught the fundamental concepts of programming such as loops, arrays, functions, classes, and event-driven programming with the help of games developed in Flash and ActionScript. Sung et al. (2008) taught programming through game-themed assignments. Chaffin et al. (2009) used an approach similar to that of Bierre and Phelps (2004) but with emphasis on data structures and algorithms. Jiau et al. (2009) also followed a game-based approach for teaching programming and stressed on the learners programming their unique strategies to play games.

Educationists continued to use game-based approaches to teach programming in the 2010s. They used board games (Li & Watson, 2011), three-dimensional graphical games (Fowler et al., 2012), and games involving programmable robots (Kazimoglu et al., 2012). In the 2010s, the focus shifted to teaching advanced level of programming using game-based approaches. Serious games were used to teach advanced concepts of programming by Tillmann et al. (2013) and Lee et al. (2013). Malliarakis et al. (2014) used a role-play game to teach programming strategies. Corral et al. (2014) used a game-based approach for teaching object-oriented programming.

Pair and Collaborative Programming

Engineering tasks are often collaborative in nature. Once they graduate, computer science students are expected to work in teams in the industry trying to solve real-life problems (Ferran et al., 2018). The teams may be large, multisite, and diverse. To prepare students for their career, educationists have been experimenting with pair programming and collaborative programming (Echeverria et al., 2017). Students learn from one another when they write programs in a pair or in a small team (Figure 5). Students get timely feedback from their teammates who also help them to debug their programs. However, choosing the right combination of students in a pair or a team is a challenge to the instructors. There are two variants of this approach of teaching programming as follows:

- Some instructors recommend students to use IDEs that support online collaboration.
- Other instructors recommend students to use conventional tools and collaborate offline.

In the 1990s, Suzuki and Kato (1995, 1997) tried to create an active learning community where students could share techniques and tips on programming among themselves. Bruckman (1997) developed a similar networked environment for teaching programming to schoolchildren. Cockburn and Bryant (1998)

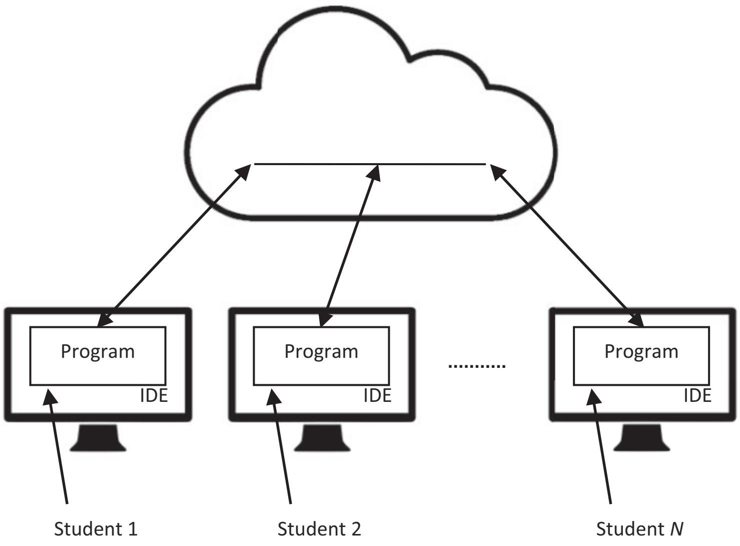


Figure 5. Use of Collaborative Learning Approach to Teach Programming.
IDE = integrated development environment.

also developed an environment that allowed students to learn programming collaboratively.

Several researchers investigated the efficacy of pair and collaborative programming in the 2000s. McNerney (2000) allowed students to learn programming by exploring ideas collaboratively with the help of tangible bricks, toys, and appliances. Williams and Upchurch (2001) observed that pair programming can help immensely in achieving an active learning experience. Pair programming lets students write code of higher quality and helps them in performing better in examination. Similar outcomes were also observed by McDowell et al. (2003). Dyba et al. (2007) studied the factors that affect the effectiveness of pair programming.

Educationists continued to research pair programming in the 2010s. Lewis (2011) observed that pair programming is highly beneficial for schoolchildren learning programming. Owolabi et al. (2013) studied different approaches of teaching programming and, in a departure from most other studies, did not find pair programming advantageous. Gomez et al. (2017) studied the effects of IDE on pair programming and recommended simpler environments.

Robot Programming

Simple robots are being used to teach programming for decades. Tangible objects such as robots attract and motivate students (Perez & Lopez, 2019).

Using robots in a course on programming makes it hands-on and less abstract (Figure 6). The students can focus on the fundamental concepts of programming without being bothered by the intricacies of the syntax of a programming language. This approach may be used to teach programming to schoolchildren as well as to teach introductory courses on programming at the undergraduate level. However, this approach is not suitable for teaching advanced courses on programming.

Mavaddat (1976) used a simple mechanical device to teach programming. The device allowed students to learn the fundamental concepts of programming by manually executing different types of statements using it. Sequential execution of statements, conditional statements, and iterative statements could be taught using the machine. In another early study, Pattis (1981) used simple robots to teach programming. Later, Lau et al. (1999) used Lego robotic kits to teach programming.

The use of robotic kits to teach programming became more common in the 2000s. Hancock (2001) taught programming by making students solve problems involving robots. Barnes (2002) used the approach to teach programming in Java, while Fagin and Merkle (2003) used it to teach students about algorithms. Sanders and Dorn (2003) developed a tool similar to the one developed by Pattis (1981) but with simpler syntax and used it to teach programming in C++ and Java. Nourbakhsh et al. (2005) and Lauwers et al. (2009) used robots for teaching programming and observed that the approach raised interest among the students. Alternatively, Bers (2010) used robots to help schoolchildren in developing computational thinking abilities.

Robots continue to be used in teaching programming. Ortiz et al. (2017) presented students with a series of programming exercises involving robots and observed significant improvement in the performance of the students at the end of the course. Lopez et al. (2017) taught programming by allowing students to program robots at a remote laboratory. Merkouris et al. (2017) also taught programming using robots and observed that the approach led to better engagement among the students.

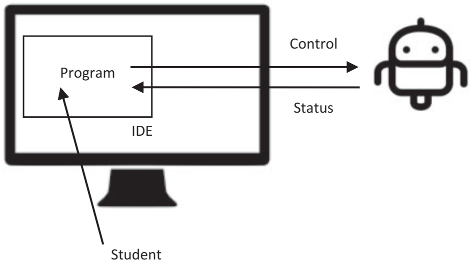


Figure 6. Use of a Simple Robot to Teach Programming.
IDE = integrated development environment.

Assessment Systems

Students attending a course on programming, introductory or otherwise, are expected to write a large number of programs. With an increasing number of students taking admission in computer science programs, it has become difficult for instructors to check and comment on all the programs written by the students (Zampirolli et al., 2018). Consequently, peer-assessment approaches and automated assessment systems are now being used in programming courses (Figure 7). Again, there are two variants of this approach as follows:

- Some instructors recommend semiautomated systems (Buyrukoglu et al., 2019) or fully automated systems (Restrepo-Calle et al., 2019; Ullah et al., 2018) to evaluate programs written by students.
- Alternatively, a few instructors ask students to evaluate programs written by their peers manually (Edwards, 2003).

Edwards (2003) used a novel approach to teach a course on programming. The students were asked to test programs written by their classmates. This helped the students to get detailed feedback on their programs and also introduced them to the concepts of software testing early in their academic program. Sitthiworachart and Joy (2003) went ahead to develop an automated assessment system that allowed students to grade programs written by one another. Brusilovsky and Sosnovsky (2005) developed an assessment system and successfully used it for four consecutive semesters to grade programs written by students in C. Higgins et al. (2005) developed a robust assessment system for grading programs written by students in object-oriented programming languages. Ahoniemi and Reinikainen (2006) developed a semiautomated assessment system that assessed programs written by students and provided feedback using English phrases. This approach is particularly helpful if a large number of

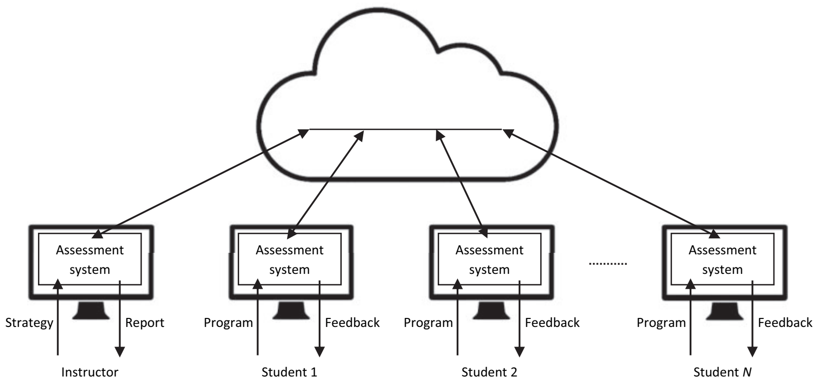


Figure 7. Use of Assessment System to Teach Programming.

students are attending the course and multiple graders are assessing their work. Suleman (2008) developed a system to assess programming assignments and integrated it with the learning management system of the university.

Recently, Law et al. (2010) developed an assessment system that checked programs written by students and evaluated their overall performance with minimal interference from the instructor. Care was taken to provide timely and relevant feedback to the students. It was observed that the use of the system motivated the students and also increased their efficacy. Contemporary assessment systems are typically implemented as web-based systems and can be accessed by the students and the instructor from anywhere and anytime (de Souza et al., 2011). Assessment systems developed in the recent years performs an in-depth analysis of the programs written by students and try to provide them detailed feedback (Powers, 2012; Veerasamy et al., 2016).

Discussion

Citation Analysis

We observed that instructors have been using different techniques for teaching courses on programming. They typically use specialized tools, often developed by themselves, to support their courses. A number of specialized tools for teaching programming have been developed so far (Table 1). These tools vary quite widely in terms of their pedagogical style and target audience. We tried to determine the influence of those tools. We found that 18 such tools have been cited 100 times or more in the literature according to Google Scholar. Scratch (Maloney et al., 2004) is undoubtedly the most researched tool for teaching programming with more than 1,000 citations. Microworld (Kelleher & Pausch, 2005) and Karel (Pattis, 1981) are also cited in the literature frequently.

Comparison of the Techniques

The five techniques used commonly for teaching programming have their own advantages and challenges (Table 2). Visual programming is typically used to introduce schoolchildren, learners from non-computer science background, and other naive learners to programming. Game-based approaches for teaching programming are advantageous because they can be personalized according to the requirements of individual learners. Initially, visual programming and game-based learning were used for teaching the basic concepts of programming only. However, now these techniques are being used to teach advanced topics as well (Corral et al., 2014; Liang et al., 2013; Mascarell, 2011; Suo, 2012). In addition, developing appropriate software tools to support these techniques requires substantial time and effort. Pair and collaborative programming, with or without special IDEs supporting online collaboration, can be used

Table 1. Tools for Teaching Programming.

Publication	Technique	Tool	Contribution	Target users	
Mavaddat (1976)	Robot Programming	Maze machine	Used a mechanical device to teach different types of statements	Undergraduate students	<10
Pattis (1981)	Robot Programming	Karel	Used simple robots to teach programming	Undergraduate students	100+
Glinert & Tanimoto (1984)	Visual Programming	Pict	A program can be written by selecting necessary constructs by clicking on appropriate icons and reordering them when necessary	Undergraduate students	100+
Bell & Lewis (1993)	Visual Programming	ChemTrains	Allows learners to write programs diagrammatically	Undergraduate students	10+
Kato & Ide (1995)	Game-based Learning	AlgoBlock	Students were made to write programs to control wrestlers in a game	Undergraduate students	10+
Pausch et al. (1995)	Visual Programming	Alice	A program is written by manipulating visual objects in a three-dimensional scene, and looks like an animated film when executed	Schoolchildren	100+
Suzuki & Kato (1995, 1997)	Pair and Collaborative Programming	AlgoBlock	Students shared notes and new programming techniques are learnt collaboratively	Schoolchildren	100+
DigGiano (1996)	Visual Programming	Chart N Art	Allows learners to write programs as a combination of diagrams and text	Students learning Lisp	10+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Freund & Roberts (1996)	Visual Programming	Thetis	Taught programming using an interactive interpreter	Undergraduate students	10+
Miyadera et al. (1996)	Visual Programming	Hichart	Taught using a visual tool and emphasized on modular programming	Undergraduate students	<10
Bruckman (1997)	Pair and Collaborative Programming	MOOSE Crossing	Students used a tool to program collaboratively	Schoolchildren	100+
Cockburn & Bryant (1998)	Pair and Collaborative Programming	Cleogo	Allowed learners to write programs collaboratively	Undergraduate students	10+
Mulholland & Watt (1998)	Visual Programming	Hank	Used a visual modeling language to teach programming	Psychology students learning to program	<10
Bagert & Calloni (1999)	Visual Programming	BACCII++	Allowed learners to diagrammatically represent a program and a tool automatically translated it into a textual form	Undergraduate students	<10
Lau et al. (1999)	Robot Programming	Lego robotic kit	Used Lego robotic kits to teach programming	Schoolchildren	10+
Davenport (2000)	Visual Programming	DELPHI	Used a visual integrated development environment	Undergraduate students	10+
McNerney (2000)	Pair and Collaborative Programming	Tangible Programming Bricks	Allowed exploring ideas collaboratively with the help of tangible bricks	Schoolchildren and undergraduate students	10+
Burnett et al. (2001)	Visual Programming	Form/3	Used a visual programming language to teach data processing using spreadsheets	Students using spreadsheet	100+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Hancock (2001)	Robot Programming	Flogo	Taught programming using robots with focus on problem solving	Undergraduate students	10+
Hartmann et al. (2001)	Visual Programming	Kara	Children learnt new concepts by programming a virtual toy	Schoolchildren	10+
Barnes (2002)	Robot Programming	Lego Mindstorms	Taught programming with focus on the concepts rather than syntax	Students learning Java	100+
Pane (2002)	Game-based Learning	HANDS	Students developed simple games	Schoolchildren	10+
Carr et al. (2003)	Visual Programming	Threadmentor	Used a thread library to explain the internal working of a program when it is executed	Undergraduate students	10+
Edwards (2003)	Assessment System	Web-Cat	Students tested programs written by one another and were introduced to the concepts of software testing	Undergraduate students	100+
Fagin & Merkle (2003)	Robot Programming	Lego Mindstorms for algorithm design and analysis*	Used robotic kits to teach algorithms	Students learning algorithm design	100+
Sanders & Dorn (2003)	Robot Programming	Jeroo	Used simple robots to teach object-oriented programming	Undergraduate students	10+
Sitthiworachart & Joy (2003)	Assessment System	Web-based peer assessment system*	Students graded programs written by one another based on parameters like correctness and readability	Undergraduate students	10+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Bierre & Phelps (2004)	Game-based Learning	Multi-User Programming Pedagogy for Enhancing Traditional Study (MUPPETS)	Students wrote programs to display and manipulate three-dimensional objects	Undergraduate students	10+
Maloney et al. (2004)	Visual Programming	Scratch	A program is written by dragging and dropping blocks representing different constructs on a canvas, and looks like a video clip when executed	Schoolchildren	1000+
Slator et al. (2004)	Visual Programming	ProgrammingLand MOOseum	Students programmed in a museum-themed virtual environment	Undergraduate students	10+
Brusilovsky & Sosnovsky (2005)	Assessment System	QuizPACK	An automated system was used to grade programs written by students	Students learning C	100+
Higgins et al. (2005)	Assessment System	Coursemaker	An automated system was used to grade programs written by students	Students learning Java	100+
Kelleher & Pausch (2005)	Visual Programming	Microworld	Taught programming using simple visual languages	Schoolchildren and undergraduate students	100+
Nourbakhsh et al. (2005); Lauwers et al. (2009)	Robot Programming	CSbots	Taught the introductory course on programming using robots	Undergraduate students	100+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Ahoniemi & Reinikainen (2006)	Assessment System	ALOHA	An automated system assessed programs written by students and provided feedback using English phrases	Undergraduate students	10+
Leutenegger & Edgington (2007)	Game-based Learning	Game-based exercise*	Taught fundamentals of programming using games	Undergraduate students	100+
Abenza et al. (2008)	Visual Programming	VisualJVM	Used a graphical integrated development environment to teach programming	Students learning Java	10+
Suleman (2008)	Assessment System	Automatic Marker	Integrated an automated assessment system with the learning management system of the university	Undergraduate students	10+
Sung et al. (2008)	Game-based Learning	Game-themed programming assignments*	Taught programming through game-themed assignments	Undergraduate students	10+
Chaffin et al. (2009)	Game-based Learning	EleMental	Introduced students to advanced concepts in programming through a game-based approach	Senior undergraduate students	10+
Jiau et al. (2009)	Game-based Learning	Game-based assignments*	Observed that learners have their unique styles of programming	Undergraduate students	10+
Bers (2010)	Robot Programming	TangibleK	Used robot programming to foster computational thinking	Kindergartners and schoolchildren	100+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Kordaki (2010)	Visual Programming	Learning Environment for programming in C using Geometrical Objects (LECGO)	Taught programming visually using geometric figures	Schoolchildren and undergraduate students	10+
Law et al. (2010)	Assessment System	Programming Assignment Assessment System (PASS)	Developed an assessment system that checked programs written by students with minimal interference from the instructor	Undergraduate students	100+
Li & Watson (2011)	Game-based Learning	Tile-based board game*	Taught programming using a board game	Schoolchildren and undergraduate students	10+
Mascarell (2011)	Visual Programming	Tool to represent concepts using unique diagrams*	Fundamental concepts of programming are explained using diagrams	Students learning C, C++ or Java	<10
de Souza et al. (2011)	Assessment System	ProgTest	Evaluated programs written by students using a Web-based system	Undergraduate students	10+
Yadin (2011)	Visual Programming	Microworld	Taught programming using simple visual languages	Students learning Python	10+
Fowler et al. (2012)	Game-based Learning	Kodu Game Lab	Taught programming using a three-dimensional graphical game	Schoolchildren	10+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Kazimoglu et al. (2012)	Game-based Learning	Program Your Robot	Taught programming using a game involving programmable robots	Undergraduate students	10+
Suo (2012)	Visual Programming	Task specific design strategy and graphical language*	Taught programming using a simplified method and attractive graphical icons	Undergraduate students	<10
Lee et al. (2013)	Game-based Learning	Gidget	Used a game-based approach to help learners in debugging their programs	Undergraduate students	10+
Liang et al. (2013)	Visual Programming	Tileland	Made learning programming simple and aesthetic	Undergraduate students	<10
Tillmann et al. (2013)	Game-based Learning	Pex4Fun	Used a game-based approach to teach advanced topics in programming	Undergraduate students	100+
Corral et al. (2014)	Game-based Learning	TUI	Taught object-oriented programming using a game-based approach	Schoolchildren	10+
Malliarakis et al. (2014)	Game-based Learning	CMX	Taught programming using a role-play game	Undergraduate students	10+
Lopez et al. (2017)	Robot Programming	Robot control and programming environment*	Students programmed robots at a remote laboratory	Not specified	10+

(continued)

Table 1. Continued.

Publication	Technique	Tool	Contribution	Target users	
Merkouris et al. (2017)	Robot Programming	Block-based visual programming environment	Taught programming using personal computers, wearable devices and robots	Not specified	<10
Ortiz et al. (2017)	Robot Programming	Mobile robot	Taught programming using exercises involving robots to motivate students	Undergraduate students	10+

* No name mentioned in original literature.

Table 2. Comparison of Techniques for Teaching Programming.

Technique	Advantage	Challenge
Visual programming	Can be used effectively to introduce schoolchildren and learners from non-computer science background to programming	Developing an integrated development environment requires substantial time and effort
Game-based learning	Can be personalized according to the requirements of individual learners	Developing the programming environment requires substantial time and effort
Pair and collaborative programming	Students get detailed and timely feedback that enables them to write code of higher quality	Finding the right composition for a pair or a team is a challenge
Robot programming	Can be used quite effectively to motivate students to learn programming	Availability of robots is a constraint
Assessment systems	Increases the scalability of the course	It is difficult to automatically assess certain aspects of programming

effectively in teaching programming courses at the undergraduate level. This technique ensures that students get detailed and timely feedback on their programs from their peers. Teaching programming using robots helps immensely in attracting students and keeping them motivated. This technique may be used to teach both schoolchildren and undergraduate students. However, the availability of working robots is often a constraint in most schools and universities. Assessment systems are typically used to check the correctness of programs written by students. However, they can also be used to assess qualities such as efficiency and readability of programs (Veerasamy et al., 2016). The use of assessment systems allows more students to enroll in programming courses.

Recommendations

We observed that the specialized tools developed for teaching computer programming help in motivating students (Dolgopolovas et al., 2018) and allow them in gaining detailed knowledge (Perez & Lopez, 2019) and performing better in exams (Echeverria et al., 2017). We recommend instructors to use such tools to enhance their courses on programming. Instructors may choose tools according to the requirements of their students. We also have some specific recommendations as follows:

- Visual programming and game-based approaches for teaching programming can enhance computational thinking ability and problem-solving skill of learners, respectively. These techniques may be used to introduce schoolchildren and other naive learners to computer programming.
- Pair and collaborative programming allows students to learn from one another and develop realistic software. This technique may be used to teach at least one course on programming at the undergraduate level.
- Robot programming can be used effectively to attract students to computer programming. Simple robots suitable for teaching computer programming may be standardized and marketed widely.
- Students of computer science attend massive open online courses (MOOCs) in large numbers, and most of them attend MOOCs related to programming (Sra & Chakraborty, 2018). Assessment systems may be integrated with such MOOCs and used for evaluating programs written by students attending them.

Conclusion

Instructors often use specialized techniques for teaching courses on computer programming, and they also develop necessary tools for the same. Such specialized techniques have been used for teaching schoolchildren as well as undergraduate students. It has been observed that suitably designed tools help in motivating students to learn programming. It has also been observed that these techniques help students to improve their computational thinking and reasoning abilities and academic performance as a whole. Some of the tools also help instructors in monitoring the progress of their students. We encourage schoolteachers and professors to use the tools and techniques discussed in this article for teaching courses on computer programming.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Pinaki Chakraborty https://orcid.org/0000-0002-2010-8022

References

Abenza, P. P. G., Olivo, A. G., & Latorre, B. L. (2008). VisualJVM: A visual tool for teaching Java technology. *IEEE Transactions on Education*, *51*(1), 86–92.

Ahoniemi, T., & Reinikainen, T. (2006). ALOHA – A grading tool for semi-automatic assessment of mass programming courses. In *Proceedings of the Sixth Baltic Sea Conference on Computing Education Research* (pp. 139–140). ACM.

Bagert, D. J., & Calloni, B. A. (1999). Teaching programming concepts using an icon-based software design tool. *IEEE Transactions on Education*, *42*(4), 14 pp.

Barnes, D. J. (2002). Teaching introductory Java through LEGO Mindstorms models. *ACM SIGCSE Bulletin*, *34*(1), 147–151.

Bell, B., & Lewis, C. (1993). ChemTrains: A language for creating behaving pictures. In *Proceedings of the IEEE Symposium on Visual Languages* (pp. 188–195). IEEE.

Bers, M. U. (2010). The TangibleK robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice*, *12*(2), Article 2.

Bierre, J., & Phelps, A. M. (2004). The use of MUPPETS in an introductory Java programming course. In *Proceedings of the Fifth Conference on Information Technology Education* (pp. 122–127). ACM.

Biju, S. M. (2013). Taking advantage of Alice to teach programming concepts. *E-Learning and Digital Media*, *10*(1), 22–29.

Bruckman, A. S. (1997). *MOOSE crossing: Construction community, and learning in a networked virtual world for kids* [PhD thesis]. Massachusetts Institute of Technology.

Brusilovsky, P., & Sosnovsky, S. (2005). Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK. *Journal on Educational Resources in Computing*, *5*(3), Article 6.

Burnett, M., Atwood, J., Djang, R., Gottfried, H., Reichwein, J., & Yang, S. (2001). Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, *11*(2), 155–206.

Buyrukoglu, S., Batmaz, F., & Lock, R. (2019). Improving marking efficiency for longer programming solutions based on a semi-automated assessment approach. *Computer Applications in Engineering Education*, *27*(3), 733–743.

Carr, S., Mayo, J., & Shene, C. K. (2003). ThreadMentor: A pedagogical tool for multi-threaded programming. *ACM Journal of Educational Resources*, *3*(1), 1–30.

Chaffin, A., Doran, K., Hicks, D., & Barnes, T. (2009). Experimental evaluation of teaching recursion in a video game. In *Proceedings of the ACM SIGGRAPH Symposium on Video Games* (pp. 79–86). ACM.

Cockburn, A., & Bryant, A. (1998). Cleogo: Collaborative and multi-metaphor programming for kids. In *Proceedings of the Third Asia Pacific Conference on Computer Human Interaction* (pp. 189–194). IEEE.

Cohen, M. (2013). Uncoupling Alice: Using Alice to teach advanced object oriented design. *ACM Inroads*, *4*(3), 82–88.

Cook, W. R. (2008). High-level problems in teaching undergraduate programming languages. *ACM SIGPLAN Notices*, *43*(11), 55–58.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107–116.

Corral, J. M. R., Balcells, A. C., Estevez, 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.

Dann, W. P., & Pausch, R. (2012). *Learning to program with Alice* (3rd ed.). Pearson.

Davenport, D. (2000). Experience using a project-based approach in an introductory programming course. *IEEE Transaction on Education*, *43*(4), 443–448.

de Souza, D. M., Maldonado, J. C., & Barbosa, E. F. (2011). ProgTest: An environment for the submission and evaluation of programming assignments based on testing activities. In *Proceedings of the Twenty-Fourth IEEE-CS Conference on Software Engineering Education and Training* (pp. 1–10). IEEE.

Deek, F. P., & McHugh, J. A. (1998). A survey and critical analysis of tools for learning programming. *Computer Science Education*, *8*(2), 130–178.

DigGiano, C. J. (1996). *Self-disclosing design tools: An incremental approach toward end-user programming* (Technical Report CU-CS-822-96). University of Colorado Boulder.

Dolgopolovas, V., Jevsikova, T., & Dagiene, V. (2018). From Android games to coding in C – An approach to motivate novice engineering students to learn programming: A case study. *Computer Applications in Engineering Education*, *26*(1), 75–90.

Dyba, T., Arisholm, E., Sjoberg, D. I. K., Hannay, J. E., & Shull, F. (2007). Are two heads better than one? On the effectiveness of pair programming. *IEEE Software*, *24*(6), 12–15.

Echeverria, L., Cobos, R., Machuca, L., & Claros, I. (2017). Using collaborative learning scenarios to teach programming to non-CS majors. *Computer Applications in Engineering Education*, *25*(5), 719–731.

Edwards, S. H. (2003). Improving student performance by evaluating how well students test their own programs. *Journal of Education Resources in Computing*, *3*(3), Article 1.

Fagin, B., & Merkle, L. (2003). Measuring the effectiveness of robots in teaching computer science. *ACM SIGCSE Bulletin*, *35*(1), 307–311.

Ferran, S., Beghelli, A., Huerta-Canepa, G., & Jensen, F. (2018). Correctness assessment of a crowdcoding project in a computer programming introductory course. *Computer Applications in Engineering Education*, *26*(1), 162–170.

Fowler, A., Fristce, T., & MacLauren, M. (2012). Kodu game lab: A programming environment. *The Computer Games Journal*, *1*(1), 17–28.

Freund, S. N., & Roberts, E. S. (1996). Thetis: An ANSI C programming environment designed for introductory use. *ACM SIGCSE Bulletin*, *28*(1), 300–304.

Glinert, E. P., & Tanimoto, S. L. (1984). Pict: An interactive graphical programming environment. *IEEE Computer*, *17*(11), 7–25.

Gomez, O. S., Aguileta, A. A., Aguilar, R. A., Ucan, J. P., Rosero, R. H., & Verdin, K. C. (2017). An empirical study on the impact of an IDE tool support in the pair and solo programming. *IEEE Access*, *5*, 9175–9187.

Hancock, C. (2001, April). *Children's understanding of process in the construction of robot behaviors* [Paper presentation]. Symposium on Varieties of Programming Experiences, Seattle, WA, United States.

Hartmann, W., Nievergelt, J., & Reichert, R. (2001). Kara, finite state machines, and the case for programming as part of general education. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments* (pp. 135–141). IEEE.

Higgins, C. A., Gray, G., Symeonidis, P., & Tsintsifas, A. (2005). Automated assessment and experiences of teaching programming. *Journal of Educational Resources in Computing*, *5*(3), Article 5.

Jiau, H. C., Chen, J. C., & Ssu, K. (2009). Enhancing self-motivation in learning programming using game-based simulation and metrics. *IEEE Transaction on Education*, *52*(4), 555–562.

Kato, H., & Ide, A. (1995). Using a game for social setting in a learning environment: AlgoArena – A tool for learning software design. In *Proceedings of the First International Conference on Computer Support for Collaborative Learning* (pp. 195–199). ACM.

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, *9*, 522–531.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Kordaki, M. (2010). A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation. *Computers & Education*, *54*(1), 69–87.

Koulouri, T., Lauria, S., & Macredie, R. D. (2014). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education*, *14*(4), Article 26.

Kunkle, W. M., & Allen, R. B. (2016). The impact of different teaching approaches and languages on student learning of introductory programming concepts. *ACM Transactions on Computing Education*, *16*(1), Article 3.

Lau, K. W., Tan, H. K., Erwin, B. T., & Petrovic, P. (1999). Creative learning in school with LEGO(R) programmable robotics products. In *Proceedings of the Twenty-Ninth Annual Frontiers in Education Conference* (pp. 26–31). IEEE.

Lauwers, T., Nourbakhsh, I., & Hamner, E. (2009). CSbots: Design and deployment of a robot designed for the CS1 classroom. *ACM SIGCSE Bulletin*, *41*(1), 428–432.

Law, K. M. Y., Lee, V. C. S., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, *55*(1), 218–228.

Lee, M. J., Ko, A. J., & Kwan, I. (2013). In-game assessments increase novice programmers' engagement and level completion speed. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 153–160). ACM.

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. *ACM SIGCSE Bulletin*, *39*(1), 115–118.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education*, *21*(2), 105–134.

Li, F. W. B., & Watson, C. (2011). Game-based concept visualization for learning programming. In *Proceedings of the Third International ACM Workshop on Multimedia Technologies for Distance Learning* (pp. 37–42). ACM.

Liang, H. N., Fleming, C., Morey, J., Sedig, K., & Man, K. L. (2013). Students' perception on the use of visual tilings to support their learning of programming concepts. In *Proceedings of the IEEE International Conference on Teaching, Assessment and Learning for Engineering* (pp. 121–126). IEEE.

Liu, X. (2014). Reform and practice in teaching data structures. In Zheng, D. (Ed.), *Proceedings of the International Conference on Education Management and Management Science* (pp. 2013–2016). CRC Press.

Lopez, M. S., Gomes, I. P., Trindade, R. M. P., Silva, A. F., & Lima, A. C. C. (2017). Web environment for programming and control of a mobile robot in a remote laboratory. *IEEE Transaction on Learning Technologies*, *10*(4), 526–531.

Malan, D. J., & Leitner, H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, *39*(1), 223–227.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2014). CMX: Implementing an MMORPG for learning programming. In Busch, C. (Ed.), *Proceedings of the Eighth European Conference on Games Based Learning* (pp. 346–355). Academic Conferences and Publishing.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: A sneak preview. In *Proceedings of the Second International Conference on Creating, Connecting, and Collaborating Through Computing* (pp. 104–109). IEEE.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education*, *10*(4), 1–15.

Martins, V. F., de Almeida Souza Concilio, I., & de Paiva Guimaraes, M. (2018). Problem based learning associated to the development of games for programming teaching. *Computer Applications in Engineering Education*, *26*(5), 1577–1589.

Mascarell, J. B. (2011). Visual help to learn programming. *ACM Inroads*, *2*(4), 42–48.

Mavaddat, F. (1976). An experiment in teaching programming languages. *ACM SIGCSE Bulletin*, *8*(2), 45–59.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In *Proceedings of the Twenty-Fifth International Conference on Software Engineering* (pp. 602–607). IEEE.

McNerney, T. S. (2000). *Tangible programming bricks: An approach to making programming accessible to everyone* [Master's thesis]. Massachusetts Institute of Technology.

Merkouris, A., Chorianopoulos, K., & Kameas, A. (2017). Teaching programming in secondary education through embodied computing platforms: Robotics and wearables. *ACM Transaction on Computing Education*, *17*(2), 9–22.

Miyadera, Y., Tsuchiya, A., Yaku, T., & Konya, H. (1996). Network-based programming language education environment based on a modular program diagram. In *Proceedings of the IEEE International Conference on Multimedia Engineering Education* (pp. 425–435). IEEE.

Mulholland, P., & Watt, S. (1998). Hank: A friendly cognitive modelling language for psychology students. In *Proceedings of the IEEE Symposium on Visual Languages* (pp. 210–216). IEEE.

Nourbakhsh, I., Crowley, K., Bhave, A., Hamner, E., Hsium, T., Perez-Bergquist, A., Richards, S., & Wilkinson, K. (2005). The robotic autonomy mobile robots course: Robot design, curriculum design, and educational assessment. *Autonomous Robots*, *18*(1), 103–127.

Ortiz, O., Franco, J. A. P., Garau, P. M. A., & Martin, R. H. (2017). Innovative mobile robot method: Improving the learning of programming languages in engineering degrees. *IEEE Transactions on Education*, *60*(2), 143–148.

Owolabi, J., Adedayo, O. A., Amao-Kehinde, A. O., & Olayanju, T. A. (2013). Effects of solo and pair programming instructional strategies on students' academic achievement in visual-basic.net computer programming language. *GSTF Journal on Computing*, *3*(3), 108–111.

Pane, J. F. (2002). *A programming system for children that is designed for usability* (Technical Report CMU-CS-02-127). Carnegie Mellon University.

Pattis, R. E. (1981). *Karel the robot: A gentle introduction to the art of programming*. Wiley.

Pausch, R., Burnette, T., Capeheart, A. C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., & White, J. (1995). Alice: Rapid prototyping for virtual reality. *IEEE Computer Graphics and Applications*, *15*(3), 8–11.

Perez, E. S., & Lopez, F. J. (2019). An ultra-low cost line follower robot as educational tool for teaching programming and circuit's foundations. *Computer Applications in Engineering Education*, *27*(2), 288–302.

Peteranetz, M. S., Flanigan, A. E., Shell, D. F., & Soh, L. (2018). Helping engineering students learn in introductory computer science (CS1) using computational creativity exercises (CCEs). *IEEE Transactions on Education*, *61*(3), 195–203.

Powers, D. M. W. (2012). Innovative instruction, assessment & evaluation in programming language concepts. In *Proceedings of the IEEE International Conference on Information Science and Technology* (pp. 512–519). IEEE.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

Restrepo-Calle, F., Echeverry, J. J. R., & Gonzalez, F. A. (2019). Continuous assessment in a computer programming course supported by a software tool. *Computer Applications in Engineering Education*, *27*(1), 80–89.

Sanders, D., & Dorn, B. (2003). Jeroo: A tool for introducing object-oriented programming. *ACM SIGCSE Bulletin*, *35*(1), 201–204.

Sitthiworachart, J., & Joy, M. (2003). Web-based peer assessment in learning computer programming. In *Proceedings of the Third IEEE International Conference on Advanced Learning Technologies* (pp. 180–184). IEEE.

Slator, B. M., Hill, C., & Val, D. D. (2004). Teaching computer science with virtual worlds. *IEEE Transactions on Education*, *47*(2), 269–275.

Sra, P., & Chakraborty, P. (2018). Opinion of computer science instructors and students on MOOCs in an Indian university. *Journal of Educational Technology Systems*, *47*(2), 205–212.

Suleman, H. (2008). Automatic marking with Sakai. In *Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries* (pp. 229–236). ACM.

Sung, K., Panitz, M., Wallace, S., Anderson, R., & Nordlinger, J. (2008). Game-themed programming assignments: The faculty perspective. *ACM SIGCSE Bulletin*, *40*(1), 300–304.

Suo, X. (2012). Toward more effective strategies in teaching programming for novice students. In *Proceedings of the IEEE International Conference on Teaching, Assessment, and Learning for Engineering* (pp. T2A1–T2A3). IEEE.

Suzuki, H., & Kato, H. (1995). Interaction-level support for collaborative learning: AlgoBlock – An open programming language. In *Proceedings of the First International Conference on Computer Support for Collaborative Learning* (pp. 349–355). ACM.

Suzuki, H., & Kato, H. (1997). Identity formation/transformation as the process of collaborative learning through AlgoArena. In *Proceedings of the Second International Conference on Computer Support for Collaborative Learning* (pp. 280–288). ACM.

Tillmann, N., Halleux, J., Xie, T., Gulwani, S., & Bishop, J. (2013). Teaching and learning programming and software engineering via interactive gaming. In *Proceedings of the Thirty-Fifth International Conference on Software Engineering* (pp. 1117–1126). IEEE.

Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A., & Saleem, F. (2018). The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, *26*(6), 2328–2341.

Veerasamy, A. K., D'Souza, D., & Laakso, M.-J. (2016). Identifying novice student programming misconceptions and errors from summative assessments. *Journal of Educational Technology Systems*, *45*(1), 50–73.

Wang, Y., Hill, K. J., & Foley, E. C. (2017). Computer programming with Python for industrial and systems engineers: Perspectives from an instructor and students. *Computer Applications in Engineering Education*, *25*(5), 800–811.

Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. *ACM SIGCSE Bulletin*, *33*(1), 327–331.

Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with Scratch in CS1. *ACM SIGCSE Bulletin*, *41*(1), 2–3.

Yadin, A. (2011). Reducing the dropout rates in an introductory programming course. *ACM Inroads*, *2*(4), 71–76.

Yuan, H.-Y., Dai, J.-G., & Peng, J. (2015). C language programming course reform and practice of teaching. In *Proceedings of the Conference on Education and Teaching in Colleges and Universities* (pp. 92–95). Atlantis Press.

Zampirolli, F. A., Goya, D., Pimentel, E. P., & Kobayashi, G. (2018). Evaluation process for an introductory programming course using blended learning in engineering education. *Computer Applications in Engineering Education*, *26*(6), 2210–2222.

Author Biographies

Kanika (BTech, Maharishi Dayanand University; and MTech, Banasthali Vidyapith) is a senior research fellow at Netaji Subhas University of Technology. She is working on developing educational tools, e-learning models and improving the quality of engineering pedagogy.

Shampa Chakraverty (BE, University of Delhi; MTech, Indian Institute of Technology Delhi; and PhD, University of Delhi) is a professor at Netaji Subhas University of Technology. Her area of research includes hardware-software co-design, analysis of sentiment, emotion and human language, and e-learning and engineering pedagogy.

Pinaki Chakraborty (BTech, Indraprastha University; and MTech and PhD, Jawaharlal Nehru University) is an assistant professor at Netaji Subhas University of Technology. His area of research includes compiler construction, educational software and effects of computers on human behavior.