Learning to Code and Collaborate in a Web Environment

Igor Škorić[1(✉)], Tihomir Orehovački[1], and Marina Ivašić Kos[2]

[1] Faculty of Informatics, Juraj Dobrila University of Pula, Zagrebačka 30, 52100 Pula, Croatia
{igor.skoric,tihomir.orehovacki}@unipu.hr
[2] Department of Informatics, University of Rijeka, Radmile Matejčić 2, 51000 Rijeka, Croatia
marinai@inf.uniri.hr

Abstract. Programming is a core skill that all computer science students should adopt, but mastering that skill is a demanding task. Educational institutions must find a way to alleviate problems associated with learning programming and to offer a service to an increased number of applicants. Part of these efforts is the use of Web tools in acquiring programming skills. The Web is a social platform and is designed to promote communication, collaboration, and sharing. Use of these tools in teaching programming prepares students for work in a distributed work environment but also opens up a possibility of improving learning process through new forms of interaction between students and lecturers. The aim of this paper is to provide an overview of Web programming tools with an emphasis on collaborative editors and discuss challenges they are addressing as well as possibilities of their application in the learning environment. As a follow up, a new taxonomy of programming learning tools is proposed to facilitate the comparison of these tools and selection of a suitable one for the particular learning activity.

Keywords: Learning programming · Collaborative learning
Web collaborative tools

1 Introduction

Programming forms the core of Computer Science (CS) education. It is essential that every CS student acquires programming skills. However, mastering programming courses is a difficult task for many students which is the reason why they have a low pass and high dropout rates [1, 2]. Lahtinen et al. [3] justified the set forth with the fact that programming contains numerous complex and abstract concepts. The syntax of standard programming languages is often complex and not well adapted to education [4]. Even if they master syntax of programming language, students do not know how to organize commands into a meaningful program [5]. Ben-Ari [6] points out that student often do not have an effective mental model of computers and do not understand the consequences of the execution of the commands or current state of the program. Some other factors (e.g. student motivation [7]) can also significantly affect their success in learning programming. Sorva et al. [8] summed up the students' difficulties and listed five major challenges that a student must overcome to master programming: static perceptions of programming, difficulties understanding the computer, misconceptions about fundamental programming constructs, and struggles with tracing of program state. These problems inspired several decades of research into the learning and teaching of programming [9]. These studies cover a wide range of topics. One of the most common topics in CS education research are programming tools [9, 10]. They represent an essential element that allows different theories, models, and techniques to be successfully applied in programming education and is therefore understandable why programming tools are important. Since the standard programming tools are not tailored to education, special tools have been developed for this purpose. Researchers have built numerous tools with the intention to facilitate teaching and learning programming [11].

The remainder of the paper is structured as follows. Section 2 describes some of the programming learning tools and gives an overview of existing taxonomies of such tools. Section 3 describes the evolution of these tools and explains why the existing taxonomy has become insufficient. Section 4 introduces and explains the new taxonomy of collaborative Web learning tools for programming. Conclusions are presented in Sect. 5.

2 Learning Programming Tools Taxonomy

The complexity of standard programming languages is one among many problems associated with programming learning all the tools for learning programming are trying to solve. The LOGO program environment [12] was developed in 1967 specifically for educational purposes. It was followed by a family of similar novice programming environments [13]. Using a series of commands, a student could draw something with a turtle graphics. In that respect, the problem that the program solved was closer and more understandable to the student. The idea that a student writes a program that solves familiar problem (like drawing rectangle) is also elaborated in programming tool ‘Karel the Robot’ [14]. This tool combines the simplified programming language and the environment in which the student program manages the robot movements. It was followed by several similar tools, such as ‘Karel J Robot’ [15] or Guido van Robot [16]. This type of tool that uses a physical metaphor to reduce the distance between the student’s mental models and program concepts is commonly referred to as a micro-world [17]. The use of visual metaphors to display certain programming concepts is a technique applied in a number of programming learning tools, such as BlueJ [18], which visualizes dynamic elements of a program (variables, objects, etc.) with simple graphical symbols. Some tools use virtual worlds in which objects are created and manipulated with programming. An example of such tools is Alice [19]. Another interesting tool for learning programming is Scratch [20]. It is designed to help children learn coding, and computational thinking. With Scratch, pupils can create interactive stories, games, and animations, and programming is done by matching different pieces of the puzzle that represent commands. Verificator [21] is a tool that prevents students from making errors and helps them to learn language syntax and adopt good programming habits. The tool forces students to properly shape the program structure, to regularly check if the program contains errors and facilitates their correction. For years, researchers have created a multitude of various tools to help students learn programming [11]. Their taxonomy is briefly explained in the following sub-section.

2.1 Taxonomy of Learning Programming Tools

To organize a wide variety of tools that can help in the acquisition of programming skills we have to somehow categorize them. In survey of tools for learning programming [22] tools are classified into the following four groups:

- *Programming environments* - Allow students to experiment with specific features of programming language and are used in program construction, compilation, testing and debugging.
- *Debugging aids* - Used by programmers to test programs, observe program behavior during execution, detect, and correct errors.
- *Intelligent tutoring systems* - Allow access to tutoring and testing material, offer adaptive instruction, analyze student responses, determine correctness, and provide feedback and advice based on stored expert knowledge.
- *Intelligent programming environments* - Combine features of intelligent tutoring systems with tools used in problem-solving and program development process.

In the survey of literature on the teaching of introductory programming [23] authors have pointed out other categories:

- *Visualization tools* - Show certain aspects of program dynamics through visual metaphors.
- *Automated rating tools* - Allow students to submit their programs, and give them a rating. These tools most commonly use pre-prepared test cases to check whether the program runs correctly.
- *Programming environments* - Different form of integrated development environment specifically adapted to the educational context
- *Other tools* - all other tools that do not belong to the previously described group

Although this taxonomy clearly classifies tools into different groups, it is not sufficiently detailed. Programming learning tools are created with a clear goal to facilitate learning a certain aspect of programming using a specific approach. In that context, when we want to select a particular tool, it is important to understand which problem it solves and in what manner. In order to understand the purpose and nature of the programming tool extensive taxonomy was developed by Kelleher and Pausch [11]. In their taxonomy, tools are categorized in two main groups: teaching systems and empowering systems. The first group consists of "systems that attempt to teach programming for its own sake" and second group of "those that attempt to support the use of programming in pursuit of another goal" [11]. Each group is further decomposed by the primary aspect of programming that the system attempts to simplify (e.g. expressing programs). In addition, each aspect is parted into a sub-problem that it solves, and each sub-problem is decomposed into the techniques by which it is solved. Each tool can appear in the taxonomy only once, based on its primary goal. This can lead to a situation where a tool that is based on solving multiple problems is only in one group, or that two similar tools are placed in different groups.

Some studies in the field related to only one set of programming tools. In this context, more authors explored the use of visualization tools in learning programming. Under the influence of the study [24] which links the success of using visualization tools and student engagement, taxonomy with five levels describing the extent of students' engagement when using visualization tools was introduced [25]. The aforementioned taxonomy describes the following levels of engagement: no viewing, viewing, responding, changing, constructing, and presenting. Study [26] presented extended engagement taxonomy (EET) in which four additional levels (controlled viewing, entering input, and reviewing) were introduced. Building on the set forth research, study [27] developed two-dimensional engagement taxonomy (2DET) that relates the direct engagement that the student has with visualization and content ownership. Levels in engagement dimension are similar to those in the EET, and levels in ownership dimension are: given content, own cases, modified content, and own content. The taxonomy was used as a classification tool for describing the visualization tools.

In [28] authors proposed three-dimensional taxonomy of Web 2.0 applications with educational potential. Proposed dimensions are: type (wiki, blog, microblog, social network…), function (collaboration, sharing, communication, knowledge organization, learning support, and artifacts integration) and cognitive processes (remembering, understanding, applying, analyzing, evaluating and creating). Although it is intended for education in general, it can be applied to learning programming as well.

3 Evolution of Learning Programming Tools

Difficulties that novices have with fundamental programming concepts encourage further research in CS education area. New studies are looking for innovative approaches, techniques, and tools that will alleviate learning. Some of the research strategies show the potential to improve the learning outcomes of programming. For instance, study [29] listed pair programming, peer teaching, and media computing as examples of such successful approaches. They have concluded that these three approaches help students to learn and retain knowledge and that their combination improves the effect. In [30] authors explored the effects of thirteen different approaches to teaching programming. Based on the analysis of primary studies, they concluded that pedagogic interventions based on collaborative learning (cooperative learning, team-based learning, and paired programming) have shown biggest improvement in learning outcomes. Research [31] found that the simultaneous use of three techniques of active learning (media computation, pair programming, and peer instruction) significantly improves students' results in the introductory programming course. Such results have contributed to the broad acceptance of collaborative techniques in learning programming, and the adoption of these techniques has created a need for tools that comply with this way of learning. Another important factor influencing the development of the learning tools is technology. The appearance of the Web 2.0 with Ajax technology, HTML 5 and JavaScript enabled the emergence of a new platform. Web environment offers many advantages over desktop [32]: unique interface always available from anywhere, instant collaboration, and easy integration with other services. Users do not need to worry about installing, maintaining, and updating. Today, we can find numerous development tools as Web services. Some of these tools are not created with the idea of being used in education. For example, CoderPad[1] was created for conducting programming interviews but can be used for real-time collaborative programming in education. Some programming environments such as CodePen[2] or repl.it[3] have no primary educational goals, and some as Codewars[4] are made solely for educational use. Some of these tools are simple such as Ideone[5], and some are complex integrated Web-based development interfaces such as Codeanywhere[6] or Cloud9[7]. These products do not have all the capabilities of their desktop counterparts but support complete process of software development on the Web. This trend of adding collaborative elements and a gradual transition to the Web platform is followed by all other programming tools. Different tools use different approaches. Codeboard[8] is a combination of Web-based IDE with elements of learning management system. CodingBat[9] besides the Web environment also includes prepared problems on which students can practice. Code Hunt[10] is a combination of program environment and video game.

Web is designed to enable communication, collaboration, and sharing. For tools in such an environment, it is natural to contain the social features. Development of distance learning and the popularity of Massively Open Online Courses (MOOCs) also contributed to the popularity of the Web. Collaboration and feeling of community are important for MOOCs to be effective [33]. Collaborative functionalities of a tool are also desirable in face-to-face learning. They allow a smaller number of teaching staff to provide education to a larger number of users.

The integrated development interface (IDE) has also changed considerably over time. Different integrated development environments have different capabilities and different toolkits, but they all contain editor, compiler, and debugger. That set of tools represents a minimum which is sufficient to a programmer to create a program. At the beginning of IDE evolution, new features were added that facilitated the programmer's work (e.g. coloring the code). All that time, the main purpose of IDE was to improve individual performance of a programmer. As software development over time grown complex, it became less the result of an individual's work and more frequently the outcome of the work of the whole team of specialists.

Collaboration is today a key element in software development because it significantly contributes to the project's performance [34] and development team members spend a

[1] https://coderpad.io/.
[2] https://codepen.io/#.
[3] https://repl.it/.
[4] https://www.codewars.com/.
[5] https://ideone.com/.
[6] https://codeanywhere.com.
[7] https://c9.io.
[8] https://codeboard.io/.
[9] http://codingbat.com/java.
[10] https://www.codehunt.com/.

significant amount of time on employing it. Moreover, nowadays it is not uncommon that the team members are dislocated. It is about a distributed software development where the challenge of achieving successful collaboration among team members is difficult. To make this collaboration among members of the development team as easy as possible, numerous tools are used. Some of these tools are common, widely accepted communication tools such as e-mail or instant messaging (IM), while other tools are specifically created to support software development such as the version control systems (VCS). In time, collaboration and communication tools have been embedded in integrated development environments such as Visual Studio[11]. The integration of collaborative tools with development tools allows developers to stay all the time in the same cognitive context (as they do not leave the environment). Such tools designed to improve the overall development team's performance are called collaborative development environments (CDEs) [35]. Today, almost all software development tools have collaborative functionality. IDEs created for professionals are often not suitable for learning as the amount of their functionalities can easily overwhelm the student. Learning tools must be easy to use and understandable. Their characteristics must be carefully chosen to match the pedagogical approach and type of education. With an objective to address this issue, we proposed a novel taxonomy that will be presented in following section.

4 Taxonomy of Collaborative Tools for Learning Programming

Programming education goes through the constant changes and tools we use in that respect follow these changes. On the one hand, the popularization of new pedagogical techniques (collaboration, peer learning, etc.) and on the other hand technological progress, led to the emergence of new tools drawing on Web 2.0 platform. Active and collaborative learning techniques encouraged the development of tools that facilitate student engagement in the learning process, their cooperation, and mutual communication. The Web platform facilitates this transformation. Novel tools should respect the nature of the Web environment and should therefore include sharing, collaboration, and communication functionalities. All the aforementioned challenges the appropriateness of using existing taxonomies for that kind of tools.

4.1 Can We Use Old Taxonomy for New Tools?

Taxonomy is a system of classification used to organize concepts into a framework for discussion or analysis. With taxonomy we can name, describe, and classify concepts on the basis of shared characteristics. If the result of our work is the taxonomy of programming learning tools, it should classify these tools by the properties that are essential to us for use of these tools. Taking into account that no single learning programming tool can address all tasks, taxonomy should help us to choose the right tool for our problem.

For example, we will take the LogoBlocks program [36], which was in a taxonomy developed by Kelleher and Pausch [11] classified in a group of tools that students can

[11] www.visualstudio.com.

use to construct programs by using objects. Let us imagine a similar Web tool that, besides the described functionality, also allows students to share the results of their work with others, and in which programmer can chat with other colleagues and other students can watch how he creates a program through a shared screen. This new tool in taxonomy introduced by Kelleher and Pausch will be in the same group as LegoBlocks. Two tools that differ significantly would be placed in the same group. Although described taxonomy has well-defined groups, due to the limitation that the tool can be in only one group, it is inadequate to describe them completely. The Web tool allows students to collaborate, and interaction is an important component of the successful collaboration. Since research [24] demonstrated that student engagement levels are positively correlated with the amount of interaction, we must also consider it as dimension. Kelleher and Pausch omitted that element in their taxonomy but dimension of engagement is part of visualization tools taxonomies [25–27]. The next important factor to take into account is the nature of communication that can be achieved through that tool. The nature of communication is complex because it can be, on one hand, synchronous or asynchronous, as well as visual, vocal or textual, on the other one. In that respect, the same tool can support many different forms of communication. Depending on whether we plan face-to-face or distant learning, synchronous or asynchronous communication, we will select a tool that supports communication that is appropriate for the planned activity type.

4.2 Proposed Taxonomy

Since different types of tools can have completely different requirements, we will only limit our study to Web based integrated development environments. In taxonomy developed by Pears et al. [23] these tools belong to the group of programming environments. They are the core tools for learning programming and most of the time in CS courses students are working with such tools. The purpose of this taxonomy is to classify these tools in a way that enables the lecturer to choose the optimal tool for a type of pedagogical activity (s)he plans to use in the course. We assume that some of the collaborative learning techniques will be used as well since in that manner the advantages of these Web tools can best come to the fore.

The proposed taxonomy (shown in Table 1.) consists of three dimensions. The first one is the type of help which describes how a tool assists the student to create a program. This dimension corresponds to the fourth column in the taxonomy developed by Kelleher and Pausch [11]. As one tool can assist the student in several ways, it can be assigned to several types of help. The lecturer can choose the tool based on the help that is the most suitable for students. The second dimension is communication which describes the types of communication students can achieve through the tool.

With this dimension, the teacher with respect to the type of planned activity (face to face, distant, synchronous, or asynchronous) selects the tool that best suits his/her needs. When choosing a communication method, students' preferences may also be considered.

Table 1. Taxonomy of Web programming environments

Dimension	Values
Type of help	Simplify the Language Prevent Syntax Errors Construct Programs Using Objects Create Programs Using Interface Actions Provide Multiple Methods for Creating Programs New Programming Models Making New Models Accessible Tracking Program Execution Make Programming Concrete Models of Program Execution Solve Problems by Positioning Objects Solve Problems Using Code Demonstrate Actions in the Interface Demonstrate Conditions and Actions Specify Actions Make the Language More Understandable Improve Interaction with Language Integration with Environment
Communication	Chat Video Audio Forum Blog Wiki Social network Blackboard …
Engagement	No viewing Sharing Viewing Communicating Changing

The third dimension is engagement which corresponds to the dimension of the same name in the taxonomy of visualization tools. One or more values can be included in this dimension:

- no viewing - every student can see only his/hers workspace
- sharing - every student can see only his/hers workspace, but can share his/her work with peers
- viewing - student can see workspace of other students or teacher
- communicating - student can see workspace of other students or teacher and can communicate through a tool
- constructing - a group of students can jointly create a program by using a tool

Each dimension allows us to evaluate one of the aspects of tool suitability. The first dimension allows us to assess whether the tool corresponds to the student's experience or age. Within this dimension we describe all the ways in which the tool provides assistance and this allows us to select very precisely the tool that best suits the planned activity. Beginners are geared to tools that offer more help. Some types of help are more suitable for younger students, so the 'Create Programs Using Interface Actions' option would be suitable for elementary students, and the 'Create Programs Using Interface Actions' option would be appropriate for preschool age.

The second dimension determines the way of communication. If we plan face-to-face activity, we will select tools that support asynchronous communication. In the case of distance learning depending on the planned type of activity, we will select the appropriate type of communication. It can appear that a tool does not incorporate communication functionality but still can be used in collaborative learning settings. In this case, the tool is limited to face-to-face teaching where communication among students is carried out in the form of verbal interaction. The third dimension determines the level of engagement of students we want to achieve during our activity. Let's say we want to inspect the code with the students. If we want the lecturer to show and explain the code in the classroom, and students should just listen, we will select a tool with 'no viewing' attribute. If we want students to download the program code on their computers, the tool must support the 'sharing' option. If we carry out the same activity in distance learning environment, the tool must support the 'viewing' option, so students can see code on their computers. If we want students to change the code being viewed then the level of engagement supported by the tool must be 'changing'.

Table 2 shows a description of three tools (ideone.com, CodeShare[12], and Code hunt) using the described taxonomy. From the first dimension we can see that ideone.com and

Table 2. Example of tool categorization

Tool	Dimensions		
	Type of help	Communication	Engagement
ideone.com	Prevent Syntax Errors		Sharing
CodeShare	Prevent Syntax Errors	Video Audio	Sharing Viewing Constructing
Code Hunt	Prevent Syntax Errors Construct Programs Using Objects Tracking Program Execution Solve Problems Using Code	–	No viewing

[12] https://codeshare.io/.

CodeShare are intended for similar audiences - students who do not need much help. In other two dimensions, these tools differ significantly. Ideone.com does not support any form of communication so it is only suitable for individual work or face-to-face group work. On the other hand, CodeShare supports audio and video communication that can be useful in remote learning. CodeShare also supports multiple levels of engagement, so it can be used for different types of activities (e.g. group code inspection or collaborative programming). Code hunt is as can be seen from a table intended for a completely different audience. It would probably be the most suitable for elementary school children because of the numerous elements of help. In addition, it does not contain any communication or engagement functions, so it is only suitable for individual work.

5 Conclusion

The tools for learning programming are one of the most important topics in computer science education research. The development of these tools is conditioned by two factors. The first one is the progress in the pedagogical interventions we use in education, and the second one is technological advancement. Tools are just a means to implement a particular strategy, model, technique or technology, and how new strategies, new models or techniques are emerging, and new tools are being developed as well. Previous studies show that the techniques that encourage student's engagement and their collaboration improve the learning outcomes of programming courses. This has led to the development of tools that support such forms of learning. The transition to the Web platform has further facilitated the development and increased the popularity of such tools. In order to achieve the optimum effect of education, it is necessary to carefully select the appropriate tool that best suits the chosen pedagogical technique. In this study, we have proposed taxonomy of Web programming environments whose aim is to alleviate a choice of tool that best suits the planned learning activity. The proposed taxonomy allows the classification of tools based on three simple dimensions that indicate whether the tool is suitable for a particular situation.

References

1. Simon, B., Lister, R., Fincher, S.: Multi-institutional computer science education research: a review of recent studies of novice understanding. In: 36th Annual Frontiers in Education Conference, pp. 12–17. IEEE (2009)
2. Watson, C., Li, F.W.: Failure rates in introductory programming revisited. In: Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, pp. 39–44. ACM (2014)
3. Lahtinen, E., Ala-Mutka, K., Järvinen, H.M.: A study of the difficulties of novice programmers. ACM SIGCSE Bull. **37**(3), 14–18 (2005)
4. Gomes, A., Mendes, A.J.: Learning to program-difficulties and solutions. In: International Conference on Engineering Education–ICEE (2007)
5. Winslow, L.E.: Programming pedagogy—a psychological overview. ACM SIGCSE Bull. **28**(3), 17–22 (1996)

6. Ben-Ari, M.: Constructivism in computer science education. ACM SIGCSE Bull. **30**(1), 257–261 (1998)
7. Jenkins, T., Davy, J.: Diversity and motivation in introductory programming. Innov. Teach. Learn. Inf. Comput. Sci. **1**(1), 1–9 (2002)
8. Sorva, J., Karavirta, V., Malmi, L.: A review of generic program visualization systems for introductory programming education. ACM Trans. Comput. Educ. (TOCE) **13**(4), 15 (2013)
9. Valentine, D.W.: CS educational research: a meta-analysis of SIGCSE technical symposium proceedings. ACM SIGCSE Bull. **36**(1), 255–259 (2004)
10. Sheard, J., Simon, S., Hamilton, M., Lönnberg, J.: Analysis of research into the teaching and learning of programming. In: Proceedings of the Fifth International Workshop on Computing Education Research Workshop, pp. 93–104. ACM (2009)
11. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. (CSUR) **37**(2), 83–137 (2005)
12. Feurzeig, W., Papert, S.A., Lawler, B.: Programming-languages as a conceptual framework for teaching mathematics. Interact. Learn. Environ. **19**(5), 487–501 (2011)
13. Guzdial, M.: Programming environments for novices. In: Computer Science Education Research, pp. 127–154 (2004)
14. Pattis, R.E.: Karel the Robot: A Gentle Introduction to the Art of Programming. Wiley, New York (1981)
15. Bergin, J., Stehlik, M., Roberts, J., Pattis, R.: Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java. Dream Songs, Redwood City (2005)
16. Kasurinen, J., Purmonen, M., Nikula, U.: A study of visualization in introductory programming. In: 20th Annual Psychology of Programming Interest Group Conference, PPIG (2008)
17. Xinogalos, S., Satratzemi, M., Dagdilelis, V.: An introduction to object-oriented programming with a didactic microworld: objectKarel. Comput. Educ. **47**(2), 148–171 (2006)
18. Kölling, M., Quig, B., Patterson, A., Rosenberg, J.: The BlueJ system and its pedagogy. Comput. Sci. Educ. **13**(4), 249–268 (2003)
19. Cooper, S., Dann, W., Pausch, R.: Alice: a 3-D tool for introductory programming concepts. J. Comput. Sci. Coll. **15**(5), 107–116 (2000). Alice: a 3-D tool for introductory programming concepts
20. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Kafai, Y.: Scratch: programming for all. Commun. ACM **52**(11), 60–67 (2009)
21. Radošević, D., Orehovački, T., Lovrenčić, A.: Verificator: educational tool for learning programming. Inform. Educ. **8**(2), 261–280 (2009)
22. Deek, F.P., McHugh, J.A.: A survey and critical analysis of tools for learning programming. Comput. Sci. Educ. **8**(2), 130–178 (1998)
23. Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Paterson, J.: A survey of literature on the teaching of introductory programming. ACM SIGCSE Bull. **39**(4), 204–223 (2007)
24. Hundhausen, C.D., Douglas, S.A., Stasko, J.T.: A meta-study of algorithm visualization effectiveness. J. Vis. Lang. Comput. **13**(3), 259–290 (2002)
25. Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Velázquez-Iturbide, J.Á.: Exploring the role of visualization and engagement in computer science education. ACM SIGCSE Bull. **35**(2), 131–152 (2002)
26. Myller, N., Bednarik, R., Sutinen, E., Ben-Ari, M.: Extending the engagement taxonomy: software visualization and collaborative learning. ACM Trans. Comput. Educ. (TOCE) **9**(1), 7 (2009)

27. Sorva, J., Karavirta, V., Malmi, L.: A review of generic program visualization systems for introductory programming education. ACM Trans. Comput. Educ. (TOCE) **13**(4), 15 (2013)
28. Orehovački, T., Bubaš, G., Kovačić, A.: Taxonomy of Web 2.0 applications with educational potential. In: Transformation in Teaching: Social Media Strategies in Higher Education, pp. 43–72 (2012)
29. Porter, L., Guzdial, M., McDowell, C., Simon, B.: Success in introductory programming: what works? Commun. ACM **56**(8), 34–36 (2013)
30. Vihavainen, A., Airaksinen, J., Watson, C.: A systematic review of approaches for teaching introductory programming and their influence on success. In: Proceedings of the Tenth Annual Conference on International Computing Education Research, pp. 19–26. ACM (2014)
31. Porter, L., Simon, B.: 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, pp. 165–170. ACM (2013)
32. Kats, L.C., Vogelij, R.G., Kalleberg, K.T., Visser, E.: Software development environments on the web: a research agenda. In: Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, pp. 99–116. ACM (2012)
33. Hew, K.F., Cheung, W.S.: Students' and instructors' use of massive open online courses (MOOCs): motivations and challenges. Educ. Res. Rev. **12**, 45–58 (2014)
34. Cook, C.L.R.: Towards computer-supported collaborative software engineering (2007)
35. Booch, G., Brown, A.W.: Collaborative development environments. Adv. Comput. **59**(1), 1–27 (2003)
36. Begel, A.: LogoBlocks: A Graphical Programming Language for Interacting with the World, pp. 62–64. Electrical Engineering and Computer Science Department, MIT, Boston (1996)