Online Tools to Support Novice Programming: A Systematic Review

Tze Ying Sim
Center for American Education,
Sunway University
Bandar Sunway, 47500 Malaysia
Email: tzeyings@sunway.edu.my

Sian Lun Lau
Dept. of Computing and Information Systems,
School of Science and Technology,
Sunway University
Bandar Sunway, 47500 Malaysia
Email: sianlunl@sunway.edu.my

***Abstract*—Novice programming is a challenging subject to both the students and the educators. A novice programmer is required to acquire new knowledge to solve a problem and propose a solution systematically. This is followed by constructing the solution in a development environment that they are unfamiliar with. This research looks at the challenges faced by a novice programmer and the online methods that are popular to assist the students. Online block programming is a popular option. One of the software that had been implemented in the various research project is Scratch. From the reviewed research, it shows that the trend is moving towards an intelligent tutoring system, where students can have personalized engagement for their learning experience. This paper presents a systematic review conducted using the keywords "novice programming", "introductory", "CS1", "difficulties", "challenges", and "threshold concepts". From the review conducted, it is observed that most of the work is carried out to ease the implementation of the solution through an integrated development environment, and block programming. On the support for instructors, the discussion on curriculum and challenges in CS1 tops the chart. This is followed by active learning through online tools.**

***Keywords*—educational technology, computer science education, engineering education**

1. Introduction

Novice programming is considered as the first programming subject taken by a student. This student will be referred to as a novice programmer in this paper. Programming is a challenging subject from a few dimension. Firstly, students are required to dissect a set of problem and propose solutions that might be able to solve the problem. The solution may be easy when described in human terms and words; however, it may be challenging when needed to be phrased in a way that the computer can understand and execute. In order, for the students to implement the solution, they will need to learn to "talk" to the computer, using a new language. Normally, the new language requires the students to be expressed using a new tool. Therefore, it is not surprising to observe that novice programming classes have about 30% to 50% failure rate [1], [2], [3]. From personal observation, the average percentage is between 40% and 50%, when considering both the withdrawal and failure rate. When only considering the failure rate, it is between 30% and 33%. Studies have shown that one of the most effective ways to have a higher passing rate is to have a small class where discussion towards the different effort to teach and learn programming can take place [3], [4]. However, having small class size is challenging as universities continue to cut on resources, and the class size continues to grow.

This paper will first look into the literature review concerning challenges in novice programming. This will then be followed by the method used for the literature review. The challenges are then categorized. Results and discussion will present the prominent findings from this literature review that are related to online tools. One of the main areas of interest for the most researcher is the support for students through block programming. There are a few online tools available. Next, the work on using online technology to overcome the challenge of engagement in big class size will be discussed. Finally, the conclusion and future work will be presented.

2. Literature Review

2.1. Challenges in Novice Programming

All of the papers reviewed suggest that students need to solve problems in a novice programming classes. They need to then implement the proposed solution in a development environment. There are three main technical challenges to a novice programming course. The challenges are listed below:

1) Propose a solution to the problem,
2) Construct the solution in a formal manner
3) Implement the solution using a new tool (i.e. the development environment).

In order to propose the solution and construct the solution in a formal order, the suitable steps arranged in the correct order are important [5]. Having formalized the solution, the students would need to implement it in a tool that is new to them. When considering each of the challenges as steps a student needs to go through before proposing a solution, they are all considered as higher order thinking activities.

There are various methodology to support the software development life cycle (SDLC) [6]. One of the earliest methods is the waterfall model. The waterfall model was designed to develop a large system and has its drawbacks of users not able to identify all the required requirements,

and developers fail to see the effort and complication of development. However, the waterfall model had been the basic model for other development like prototyping, agile programming, and rapid application development. For the purpose of this paper, the general term SDLC will be used to refer to the development phases. The steps for development are:

1) Planning and defining requirement analysis
2) Designing the solution
3) Implementing the formalized solution
4) Testing the solution
5) Documentation and Deployment

Most novice programming classes include the planning, designing, implementing and testing phase [7]. As the projects for novice programming are small, students may repeat the implementing, and testing phase repeatedly, and when needed the design phase again.

3. Methodology

A systematic review was conducted using the keywords "novice programming", "CS1", "introductory programming", "threshold concepts", "difficulties", and "challenges". Database searched are mainly from the Association for Computing Machinery (ACM), Institute of Electrical and Electronics Engineers (IEEE). The articles were published between 1998 and 2018. The articles are then categorized according to the phases in the waterfall model as well as the pedagogy support for the instructor. When possible, a paper will be marked in one category. However, if the focus area of multiple sections, then multiple categories will be checked. The description for each of the development phase are as follows:

1) Problem abstraction and solution proposal
2) Formalizing the solution
3) Implementing the formalized solution through tools support
4) Testing the implemented solution
5) Documentation and submission of work

Apart from looking at the need of the students. The support for the instructor is also given attention. The support for instructors is divided into the following categories:

1) Active learning through tools
2) Active learning without using tools
3) Assessment tools
4) Curriculum and Threshold Concepts
5) Collaborative learning and other pedagogy ideas

The word tools in the support for the instructor overlaps with the tools in the development phase. Tools to implement the solution are applicable to both categories. For a paper that covers both pedagogy with the support of an implementation tool, then the "x" will be marked at the implementation tool column. If the tool is not an implementation tool, for example, an iPad, blended learning, or online tools, then the "x" will be marked at the pedagogy support column. If a tool is used to investigate certain ideas, but most of the discussion in the paper is not about the impact of the tools but on the idea, alongside with other methods, then the "x" will be marked at the pedagogy support. To the best possible, each paper will be assigned to the column most relevant to its content.

TABLE 1. PAPERS MAPPED TO THE DEVELOPMENT PHASE

Development Phase for CS1	References	Publication Count
Problem abstraction and solution Proposal		0 (0%)
Formalizing the solution		0 (0%)
Error Messages	[8] [9] [10] [11]	7 (20.6%)
New Programming Environment	[12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23]	20 (58.8%)
Hands On Project/ Robot	[24] [25] [26]	5 (14.7%)
Testing the implemented solution		1 (2.9%)
Documentation & Submission of Work		1 (2.9%)

TABLE 2. PAPERS MAPPED TO THE SUPPORT FOR INSTRUCTOR

Support for Instructor	References	Publication Count
Active learning through tools	[27] [28] [29] [30] [31] [32] [33] [34] [35] [2] [36] [37] [38] [39] [40] [41]	23 (24.0%)
Active learning without using tools	[42] [43] [44] [45] [46] [5]	11 (11.5%)
Assessment tools	[47] [48] [49] [50] [51] [52]	11 (11.5%)
Curriculum and Threshold Concepts	[53] [54] [7] [55] [56] [57] [58] [59]	24 (25.0%)
Other pedagogy ideas	[60] [61] [62] [63] [64] [65] [66] [3] [67] [68]	27 (28.1%)

4. Results

The papers analyzed are between 1998 and 2018. There are not so many papers returned for the year 2017 and 2018. The peak three years are between 2013 and 2015. This is followed by 2010, 2012, and 2016. When listing the references for each of the categories, only publications in the last five years, starting from 2013 are listed. 34 papers were mapped under the development phase, and another 98 were mapped under the support for instructor.

When categorizing the focus area of each reviewed paper according to the software development phase, most of the papers focused on the implementation phase. The main idea here is to simplify the process of implementation through enhanced integrated development environment (IDE), minimizing syntax errors when typing the code through block programming, and increasing students' interest through hands-on projects. Please see Table 1 for more information.

When looking into support for the lecturer, it is observed that most of the work done in a single area is in relation to the curriculum and the threshold concepts of cs1 (25.8%). This is followed by active learning through tools at 24.7%. Other pedagogy ideas that stands at 28.1% includes various methods like self-regulated learning, pair programming, social learning, mental model, etc. Please see Table 2 for more information.

5. Discussion

From the result of 34 publications that are directly related to a specific phase in the development cycle, 32 of them concern the implementation of the solution. Almost no research or support was explicitly designed to support the other phases in the development cycle, which is the main focus in novice programming environment. Novice programming environment involves the development of block programming and the enhancement of IDE by making error messages more understandable. On the other hand, as for the support for instructors, the most focused area is active learning through tools engagement. One of the trends observed in this category is the use of online tools to support the teaching and learning activities. Examples of tools include generation of questions, intelligent tutoring system, and blended learning. For the following discussion, the focus would be on tools that support block programming and intelligent tutoring system.

5.1. Novice Programming Environment

Novice programming environment covers the various initiative to make novice programming easier. In a conventional system, the student would need to translate the formalized solution to codes that need to be typed in either through an IDE or a text-based interface. This proves to be challenging as the student faces problems with the syntax [10]. By using the drag-and-drop interface in block programming, this completely eliminates the syntax error [19], [34], and students only need to concentrate on the structure and logic of the program. From the twenty articles reviewed, there were about twenty programming environment identified. However, some of the environments are not included in Figure 1 as: the authors did not manage to find the tool online for verification or the tools are developed to support learning through visualization, but not making a change in the way the program is implemented. The examples are Jeliot (http://cs.joensuu.fi/jeliot/) and Jeeroo (http://home.cc.gatech.edu/dorn/jeroo).

Firstly, the genre of tools will be discussed. Most of the tools allow the students to creatively develop their application except Gidget. Gidget is similar to "Hour of Code" where students are provided with a set of online games (exercises) to understand the fundamentals of the programming structure. Upon completion of all the required level, the level designer will be enabled where students can create their own levels and share it with other users [69]. Game development is the most popular genre. Game development motivates students participation in CS1. They are able to create, play own games, and evaluate games from their peers. The second most popular genre is narrative. Narrative allows the user to create stories. These tools are also able to create some simple games. Hence, the genre "Narrative/Game". The category application is given to tools that also supports other application apart from narrative and game.

Most of the tools identified are available for free. Only three tools have the "pay option". They are Construct2, GameMaker, and GameSalad. Even professionals can use these tools to develop games for sale. Not all of them are online tools. About 50% of the tools are available online and the other 50% requires users to download the IDE for offline development. The advantage of using the tool online is no setup is required. The student only needs to register and login via the web interface to work on their project. Automatic step by step tracing helps the students to visualize the impact of each line of code. This feature is provided by Scratch. Using Scratch each execution step in highlighted in the workspace. Snap! highlights the whole block of codes that is currently active. It does not go through each of the individual lines of code (or in this case lines of blocks). As for other tools, the current active code will not be shown when the program run.

One of the debates in the curriculum of CS1 is the imperative first vs. object first concept [70]. Many institutions introduce the Object Oriented Programming (OOP) concepts even in the imperative first concept. Therefore, it will be helpful if some of the OOP concepts like Abstraction, Polymorphism, Inheritance, and Encapsulation (APIE), can be included in the tool. The tools that are marked with "support OOP" introduces the concept of class and objects or instances. The objects created can be customized values can be assigned to their properties. The objects have defined methods that can be called for execution. Greenfoot even visualizes the class diagram as part of the development interface.

When reviewing the development method, the tools are grouped into drag and drop method (also commonly known as the block-based programming) or text-based method. The former comes with the advantage of zero syntax error. There will be no worry of a missing semicolon, misplaced curly brackets, misplaced bracket, using the wrong keywords, etc. Students can drag the relevant block, and configure the parameters within the block. However, one major challenge is the industry does not program with block-based programming. Eventually, the students need to move on to text-based programming. Most of the text-based programming required for the listed tools are the programming of events for a particular object. Most of the tools which apply the drag and drop method do not support the transition to text (see Figure 1). However, there are two tools that the coding methods are labeled as drag and drop but they support the transition to text programming. This is because, the blocks are designed to look like complete programming code, with parameters that are modifiable. One unique tool is Greenfoot. Greenfoot uses frame-based programming, where the codes are typed. However, they are grouped according to blocks, for example, if-block, while-block, etc. It is not possible to drag and drop a block to part of the code that it is syntactically not supposed to be. The tool is between block-based programming and text-based programming.

From the list of tools listed, Scratch is the most popular, with seven publications. This is followed by App Inventor with 3, Alice, Game Salad, BlueJ and Raptor have 2 publications each. However, I believe the trend will change for the future. Scratch is good tool to introduce the basic concepts. This is suitable to introduce problem-solving skills to high school students, or even as a workshop for a novice programmer. A novice programmer will eventually progress on to higher level programming subject, and be developing actual applications. Therefore, it is important for the tool to support the transition to text-based programming, and the development of the various applications. Having reviewed the tools, we would pro-

Novice Programming Environments	Website	Genre	Free/ Paid?	Online Tool	Automatic Step by Step Tracing	Supports OOP	Coding Method - Drag & Drop (DD), Text Based (Text)	Supports Transition to Text Programming
Alice 2	https://www.alice.org/	Narrative/Game	Free	No	No	No	DD	No
Alice 3	https://www.alice.org/	Narrative/Game	Free	No	No	Yes	DD	Yes
AppInventor	http://appinventor.mit.edu/explore/	Application	Free	Yes	No	Yes	DD	Yes
Construct 2	https://www.construct.net/	Game Development	Free/Paid	Yes	No	Yes	Text	No
GameMaker	https://www.yoyogames.com/gamemaker	Game Development	Free/Paid	No	No	Yes	Text	Yes
GameSalad	https://edu.gamesalad.com/	Game Development	Free/Paid	No	No	Yes	DD	No
Gidget	https://www.helpgidget.org/	Game Development	Free	Yes	No	No	Text	Yes
Greenfoot (continuation of Blue J)	https://www.greenfoot.org/door	Application	Free	No	No	Yes	Frame	Yes
Looking Glass	https://lookingglass.wustl.edu/	Narrative/Game	Free	No	No	Yes	DD	No
Pencilcode	https://pencilcode.net/	Narrative/Game	Free	Yes	No	No	DD	No
Raptor	https://raptor.martincarlisle.com/	Application	Free	No	Yes	Yes	DD & Text	No
Scratch	https://scratch.mit.edu/	Narrative/Game	Free	Yes	Yes	No	DD	No
Snap!	https://snap.berkeley.edu/	Application	Free	Yes	No	Yes	DD	No
Tululoo	http://www.tululoo.com/	Game Development	Free	No	No	Yes	Text	Yes

Figure 1. Development tools reviewed

pose App Inventor and Greenfoot. Among the two, Greenfoot will be the preferred choice, as it allows students to type the code while limiting the potential syntax errors.

5.2. Intelligent Programming Tutors

An Intelligent Programming Tutor (IPT) is defined as an intelligent tutoring system that is specifically created for programming education [34]. The purpose of the creation of such tools is to provide computer-assisted learning with adaptive or personalized features. IPTs can be a useful tool in novice programming education because a learner can learn at his own pace in a one-to-one tutoring setting and still get personalized assistance offered by an IPT with similar outcome or benefits as compared to human tutoring [71].

Pillay [72] analyzed several IPTs in 2005 and developed a generic architecture for the development of IPTs. The analysis of IPTs mentioned in [72] identified the five main functions, which include presentation and explanation of programming concepts, provision of different programming problems for learning purposes, assistance to help students to solve various programming problems, assessment of students' programming quality as well as assistance to debug programs for semantic errors.

iSnap [20] is an IPT that offers on-demand, next-step hints to novice programming students. It is an extension to the Snap! Novice Programming Environment (NPE). The hints are generated using Contextual Tree Decomposition (CTD), a data-driven algorithm. Students' solutions are recorded and used as data to provide recommendations and hints for future students' attempts. On-demand hints will only be provided if a student selects to ask for a hint via a Help button. The pilot study discovered several challenges, including students' bottom-up programming behavior, over-reliance on help and difficulty understanding the ideas behind the hints.

Hooshyar et al. [73] applied a flowchart-based IPT to improve problem-solving skills of novice programmers. The authors argue that there are three advantages: Firstly, the process of problem-solving is emphasized. Secondly, it enables learners' engagement by providing an interactive problem-solving environment. Thirdly, Bayesian Networks is used to assist the students' learning progress. 44 students tested the system and the overall experience was positive. Improvement in problem-solving skills can be observed in the outcome.

The above examples are by no means exhaustive. However, common attributes can be observed in these examples: problem-solving skills have been the common focus in these IPTs. Also, the use of an algorithm and machine learning can also help to make these tools more useful and effective in helping novice programmers to learn programming. More importantly, submitted answers and attempts are a good source of information that can be used to train the models needed in these IPTs. Particularly the weaker students who need more time and guidance, such IPTs may be helpful for them to learn at their own pace.

6. Conclusion and Future Work

The paper reviewed about 130 articles from the past 20 years and concluded that

1) most of the research in supporting novice programming focused on the implementation of new programming environments that either simplify the mechanism of solution implementation, enhance students understanding on the code through output visualization or memory visualization, or increase students' interest to the subject by creating narrative and games.
2) in order to support the instructor, various work on curriculum and pedagogy had been conducted,

especially in the area of active learning with tool support

3) in order to overcome the issue of increasing student and staff ratio, intelligent programming tutors are implemented to support the learning needs of students.

For future work, it will be interesting to see

1) new programming environment to further support other development phases like solution design, and documentation.
2) further development of the intelligent programming tutors to engage students learning

References

[1] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *ACM SIGCSE Bulletin*, vol. 39, no. 2, p. 32, 2007.

[2] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *Proceedings - Frontiers in Education Conference, FIE*, 2015.

[3] C. Watson and F. W. Li, "Failure rates in introductory programming revisited," in *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14*, 2014, pp. 39–44.

[4] X. Suo, "Session T2A Toward More Effective Strategies in Teaching Programming for Novice Students," in *IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, Hong Kong, 2012, pp. T2A1–T2A3.

[5] T. Y. Sim, "Exploration on the impact of online supported methods for novice programmers," in *2015 IEEE Conference on e-Learning, e-Management and e-Services, IC3e 2015*, 2016, pp. 158–162.

[6] V. M. Font Jr., *The Ultimate Guide to the SDLC*, 1st ed. North Carolina: A FrontLife Publication, 2012.

[7] A. Luxton-Reilly, "Learning to Program is Easy," *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*, pp. 284–289, 2016.

[8] A. Altadmri and N. C. C. Brown, "37 Million Compilations," *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, pp. 522–527, 2015.

[9] M. Amaratunga and S. Rajapakse, "An Interactive Programming Assistance Tool (iPAT) for Instructors and Novice Programmers," in *The 8th International Conference on Computer Science & Education (ICCSE 2013)*, no. Iccse, Colombo, Sri Lanka, 2013, pp. 680–684.

[10] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney, "Effective compiler error message enhancement for novice programming students," *Computer Science Education*, vol. 26, no. 2-3, pp. 148–175, 7 2016.

[11] N. C. Brown and A. Altadmri, "Investigating Novice Programming Mistakes:Educator Beliefs vs Student Data," in *Proceedings of the tenth annual conference on International computing education research - ICER '14*, 2014, pp. 43–50.

[12] J. Good and K. Howland, "Natural language and programming: Designing effective environments for novices," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2015.

[13] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? A controlled experiment on Scratch programs," in *IEEE International Conference on Program Comprehension*, vol. 2016-July, no. Section IV, 2016, pp. 1–10.

[14] Y. Hosanee and S. Panchoo, "An Enhanced Software Tool to Aid Novices in Learning Object Oriented Programming (OOP)," in *Computing, Communication and Security (ICCCS), 2015 International Conference on*, Pamplemousses, 2015.

[15] M. Ichinco and C. Kelleher, "Exploring novice programmer example use," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2015.

[16] M. Kölling, N. C. C. Brown, and A. Altadmri, "Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming," in *Proceedings of the Workshop in Primary and Secondary Computing Education*, 2015, pp. 29–38.

[17] M. Kölling and F. McKay, "Heuristic Evaluation for Novice Programming Systems," *ACM Transactions on Computing Education*, vol. 16, no. 3, p. 30, 2016.

[18] O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, "Learning computer science concepts with Scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, 9 2013.

[19] S. Papadakis, M. Kalogiannakis, V. Orfanakis, and N. Zaranis, "Novice Programming Environments. Scratch & App Inventor," *Proceedings of the 2014 Workshop on Interaction Design in Educational Environments - IDEE '14*, pp. 1–7, 2014.

[20] T. W. Price and T. Barnes, "Comparing Textual and Block Interfaces in a Novice Programming Environment," *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*, pp. 91–99, 2015.

[21] J. Robertson, "Rethinking how to teach programming to newcomers," *Communications of the ACM*, vol. 57, no. 5, pp. 18–19, 2014.

[22] D. Weintrop, "Blocks, text, and the space between: The role of representations in novice programming environments," *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, vol. 2015-Decem, no. C, pp. 301–302, 2015.

[23] T. Wyeld and Z. Barbuto, "Don't Hide the Code!: Empowering Novice and Beginner Programmers Using a HTML Game Editor," *2014 18th International Conference on Information Visualisation*, pp. 125–131, 2014.

[24] L. Major, T. Kyriacou, and P. Brereton, "The effectiveness of simulated robots for supporting the learning of introductory programming: a multi-case case study," *Computer Science Education*, vol. 24, no. 2-3, pp. 193–228, 2014.

[25] R. Mason and G. Cooper, "Mindstorms robots and the application of cognitive load theory in introductory programming," *Computer Science Education*, vol. 23, no. 4, pp. 296–314, 12 2013.

[26] M. A. Rubio, R. Romero-zaliz, C. Mañoso, and A. P. D. Madrid, "Enhancing an introductory programming course with physical computing modules," 2014.

[27] H. Amer and W. Ibrahim, "Using the iPad as a pedagogical tool to enhance the learning experince for novice programing students," in *IEEE Global Engineering Education Conference, EDUCON*, 2014.

[28] T. Arabacioglu and R. Akar-Vural, "Using facebook as a LMS?" *Turkish Online Journal of Educational Technology*, vol. 13, no. 2, pp. 202–215, 2014.

[29] M. Armoni and J. Gal-Ezer, "High school computer science education paves the way for higher education: the Israeli case," *Computer Science Education*, vol. 24, no. 2-3, pp. 101–122, 7 2014.

[30] H. Awni, A. Rekhawi, and S. S. Abu Naser, "An Intelligent Tutoring System for Learning Android Applications UI Development," *International Journal of Engineering and Information Systems*, vol. 2, no. 1, pp. 2000–0, 2018.

[31] T. Ball and B. Zorn, "Teach foundational language principles," *Communications of the ACM*, vol. 58, no. 5, pp. 30–31, 2015.

[32] M. Bower, B. Dalgarno, G. E. Kennedy, M. J. W. Lee, and J. Kenney, "Design and implementation factors in blended synchronous learning environments: Outcomes from a cross-case analysis," *Computers and Education*, vol. 86, pp. 1–17, 2015.

[33] J. Campbell, D. Horton, and M. Craig, "Factors for Success in Online CS1," in *Innovation and Technology in Computer Science Education (ITiCSE)*, 2016, pp. 320–325.

[34] T. Crow, A. Luxton-Reilly, and B. Wuensche, "Intelligent tutoring systems for programming education," *Proceedings of the 20th Australasian Computing Education Conference on - ACE '18*, pp. 53–62, 2018.

[35] M. R. Donggil Song, Eun Young Oh, "Interacting with a Conversational Agent System for Educational Puroposes in Online Courses," *22017 10th International Conference on Human System Interactions (HSI)*, pp. 3–7, 2017.

[36] Y. Iizuka, S. Enari, and K. Kinoshita, "Designing a novice programming environment for RoboCup Soccor 2D simulation," *Proceedings - 2nd IIAI International Conference on Advanced Applied Informatics, IIAI-AAI 2013*, pp. 401–402, 2013.

[37] A. S. Kim and A. J. Ko, "A Pedagogical Analysis of Online Coding Tutorials," *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE '17*, pp. 321–326, 2017.

[38] S. Kurkovsky, "Mobile game development: improving student engagement and motivation in introductory computing courses," *Computer Science Education*, vol. 23, no. 2, pp. 138–157, 6 2013.

[39] S. Matsumoto, K. Okimoto, T. Kashima, and S. Yamagishi, *Human-Computer Interaction. Theory, Design, Development and Practice*, K. M., Ed. Springer, Cham, 2016, vol. 9731.

[40] R. Matthews, H. S. Hin, and K. A. Choo, "Novice Programming Students' Perception of Learning Object," *2013 International Conference on Informatics and Creative Multimedia*, pp. 292–297, 2013.

[41] D. Mccall and M. Kölling, "Meaningful Categorisation of Novice Programmer Errors," in *IEEE Frontiers in Education Conference (FIE) Proceedings*, 2014, pp. 1–5.

[42] T. B. Bati, H. Gelderblom, and J. van Biljon, "A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa," *Computer Science Education*, vol. 24, no. 1, pp. 71–99, 1 2014.

[43] K. L. Eranki and K. M. Moudgalya, "Application of program slicing technique to improve novice programming competency in spoken tutorial workshops," *Proceedings - IEEE 6th International Conference on Technology for Education, T4E 2014*, pp. 32–35, 2014.

[44] D. Horton and M. Craig, "Drop, Fail, Pass, Continue," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, 2015, pp. 235–240.

[45] M. Ichinco, A. Zemach, and C. Kelleher, "Towards generalizing expert programmers' suggestions for novice programmers," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2013.

[46] T. Y. Sim, "My Learning Journal," 2015.

[47] J. C. Caiza and J. M. del Alamo Ramiro, "Programming Assignments Automatic Grading: Review of Tools and Implementations," *7th International Technology, Education and Development Conference*, pp. 5691–5700, 2013.

[48] A. Luxton-Reilly and A. Petersen, "The Compound Nature of Novice Programming Assessments," in *Proceedings of the Nineteenth Australasian Computing Education Conference on - ACE '17*, 2017, pp. 26–35.

[49] S. Nutbrown and C. Higgins, "Static analysis of programming exercises: Fairness, usefulness and a method for application," *Computer Science Education*, vol. 26, no. 2-3, pp. 104–128, 7 2016.

[50] A. Shargabi, S. A. Aljunid, M. Annamalai, S. M. Shuhidan, and A. M. Zin, "Tasks That Can Improve Novices' Program Comprehension," in *IEEE Conference on e-Learning, e-Management and e-Services*, Malacca, 2015, pp. 32–37.

[51] S. Shuhidan, M. Hamilton, and D. D'Souza, "A taxonomic study of novice programming summative assessment," *Conferences in Research and Practice in Information Technology Series*, vol. 95, no. Ace, pp. 147–156, 2009.

[52] J. Vahrenhold and W. Paul, "Developing and validating test items for first-year computer science courses," *Computer Science Education*, vol. 24, no. 4, pp. 304–333, 10 2014.

[53] A. S. Alardawi and A. M. Agil, "Novice Comprehension of Object-Oriented OO Programs: An Empirical Study," in *Information Technology and Computer Applications Congress (WCITCA), 2015*, 2015.

[54] P. Alston, D. Walsh, and G. Westhead, "Uncovering Threshold Concepts in Web Development: An Instructor Perspective," *ACM Transactions on Computing Education*, vol. 15, no. 1, pp. 1–18, 2015.

[55] M. Piteira and C. Costa, "Learning Computer Programming: Study of difficulties in learning programming," *International Conference on Information Systems and Design of Communication*, pp. 75–80, 2013.

[56] K. Sanders and R. McCartney, "Threshold Concepts in Computing : Past , Present , and Future," in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, Finland, 2016, pp. 91–100.

[57] D. Shinners-Kennedy and S. A. Fincher, "Identifying threshold concepts," in *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, 2013, p. 9.

[58] C. Thevathayan and M. Hamilton, "Supporting Diverse Novice Programming Cohorts through Flexible and Incremental Visual Constructivist Pathways," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '15*, 2015, pp. 296–301.

[59] G. Walker, "A cognitive approach to threshold concepts," *Higher Education*, vol. 65, no. 2, pp. 247–263, 2013.

[60] Y. B. Kafai and Q. Burke, "The social turn in K-12 programming," *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, p. 603, 2013.

[61] M. Maleko, M. Hamilton, D. D'Souza, and F. Scholer, "Understanding and analysing novice programmer interactions in a facebook programming group," in *Proceedings - 2014 International Conference on Teaching and Learning in Computing and Engineering, LATICE 2014*, 2014, pp. 112–119.

[62] C. S. Miller, "Metonymy and reference-point errors in novice programming," *Computer Science Education*, vol. 24, no. 2-3, pp. 123–152, 7 2014.

[63] A. Mühling, "Aggregating concept map data to investigate the knowledge of beginning CS students," *Computer Science Education*, vol. 26, no. 2-3, pp. 176–191, 2016.

[64] C. Ott, A. Robins, P. Haden, and K. Shephard, "Illustrating performance indicators and course characteristics to support students self-regulated learning in CS1," *Computer Science Education*, vol. 25, no. 2, pp. 174–198, 4 2015.

[65] J. Rountree, A. Robins, and N. Rountree, "Elaborating on threshold concepts," *Computer Science Education*, vol. 23, no. 3, pp. 265–289, 9 2013.

[66] A. Tafliovich, J. Campbell, and A. Petersen, "A student perspective on prior experience in CS1," in *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, 2013, p. 239.

[67] S. Willman, R. Lindén, E. Kaila, T. Rajala, M.-J. Laakso, and T. Salakoski, "On study habits on an introductory course on programming," *Computer Science Education*, vol. 25, no. 3, pp. 276–291, 7 2015.

[68] M. Zarb and J. Hughes, "Breaking the communication barrier: guidelines to aid communication within pair programming," *Computer Science Education*, vol. 25, no. 2, pp. 120–151, 4 2015.

[69] M. J. Lee and A. J. Ko, "A Demonstration of Gidget, A Debugging Game for Computing Education !" Tech. Rep.

[70] The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society, "Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science," ACM, IEEE Computer Society, Tech. Rep., 2013.

[71] J. A. Kulik and J. D. Fletcher, "Effectiveness of Intelligent Tutoring Systems: A Meta-Analytic Review," *Review of Educational Research*, vol. 86, no. 1, pp. 42–78, 2016.

[72] N. Pillay and V. R. Jugoo, "An investigation into student characteristics affecting novice programming performance," *ACM SIGCSE Bulletin*, vol. 37, no. 4, p. 107, 2005.

[73] D. Hooshyar, R. B. Ahmad, M. Yousefi, F. D. Yusop, and S.-J. Horng, "A Flowchart-Based Intelligent Tutoring System for Improving Problem-Solving Skills of Novice Programmers," *Journal of Computer Assisted Learning*, vol. 31, no. 4, pp. 345–361, 2015.