Computing Education in K-12 Schools: A Review of the Literature

Varvara Garneli
Department of Informatics
Ionian University
Corfu, Greece
c13garn@ionio.gr

Michail N. Giannakos
Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
mgiannakos@acm.org

Konstantinos Chorianopoulos
Department of Informatics
Ionian University
Corfu, Greece,
choko@acm.org

Abstract—**During the last few years, the focus of computer science education (CSE) in primary and secondary schools (shortly K-12) have reached a significant turning point. This study reviews the published papers on the field of K-12 computing education in order to summarize the findings, guide future studies and give reflections for the major achievements in the area of CSE in K-12 schools. 47 peer-reviewed articles were collected from a systematic literature search and analyzed, based on a categorization of their main elements. Programming tools, educational context, and instructional methods are the main examined categories of this research. Results of this survey show the direction of CSE in schools research during the last years and summarized the benefits as well as the challenges. In particular, we analyzed the selected papers from the perspective of the various instructional methods aiming at introducing and enhancing learning, using several programming tools and educational context in K-12 CSE. Despite the challenges, the findings suggest that implementing computing lessons in K-12 education could be an enjoyable and effective learning experience. In addition, we suggest ways to facilitate deep learning and deal with various implications of the formal and informal education. Encouraging students to create their own projects or solve problems should be a significant part of the learning process.**

Keywords—computer science education, computer programming, programming pedagogy, educational context, programming tools, K-12 Education.

I. INTRODUCTION

The continual development of digital technology has already changed the formal and informal education in many ways. The technology based education could support and enhance learning through several computer applications [18]. Moreover, new skills related to the learners' work in a digital environment need to be developed and measured [3]. Under this perspective, exploring the role of Computer Science in a technology based educational context might be interesting. Introducing and enhancing computing concepts and skills for example could provide learners with the opportunity to be creatively engaged in various learning activities. Moreover, boosting students' interest about Computer Science Education (CSE) and programming using several extracurricular activities is an interesting perspective which seems to be very effective in promoting computer science concepts and practices or broadening CSE [34][45]. Additionally, implementing computing lessons in various ways and in a more regular basis like in the typical school environment, could enhance learning and benefit students with the development of fundamental skills like computational thinking [21] and creativity [15][16]. Computing education and especially programming could support students in many ways in a carefully designed learning setting. This perspective does not mean that all learners will become necessarily professional programmers but that they will gain useful practices and skills of the digital era.

Most of the early research on this area has focused on the various programming tools description. In addition, other approaches explore the difficulties, underline the benefits, and suggest ways to deal with the several problems related to the instruction of computer programming. Despite the multiple efforts, the recent technical, infrastructural and societal developments posit K-12 CSE research ripe for more exploration. Scholars and educators have reported a variety of outcomes from computing and programming initiative in K-12 schools; however the lack of a summarization from all these empirical studies prevents stakeholders from having a clear view of the benefits and the challenges. The purpose of this article is to provide a review of research on K-12 computing education approaches in order to summarize the findings, guide future studies and give reflections for the major achievements in the area of CSE in K-12. Particularly, we explore various approaches and their effectiveness on introducing and enhancing learning using various programming tools and educational contexts in K-12 CSE. In addition, we suggest instructional methods in order to facilitate deep learning and deal with various implications of the formal and informal education. Particularly, we explore the following research questions:

RQ1: Which are the benefits and the challenges of using diverse programming tools in K-12 computing education?

RQ2: What are the mainstream educational contexts for motivating students and improving their learning in K-12 computing education?

RQ3: What are the most common instructional practices and how are educators putting them into practice?

The rest of the paper is structured as follows: in the second section related work is presented, in the third section is described the methodology used to carry out the systematic literature review and follows a discussion on the results. Finally, we present conclusions of the reported research.

II. RELATED WORK

During the last decades, many researchers have conducted and published several studies in the field of computer programming. Developed programming languages for novices and professionals and their features, programming curricula, and teaching strategies, are some of the various perspectives of this research area.

Since the 1960's, many programming languages and environments have been developed, setting this way programming widely accessible. Kelleher and Pauschin (2005) presented their taxonomy of languages for novice programmers of all ages. According to their research [9], there are several programming tools which support various programming styles, programming constructs, code's look, or coding actions. Additionally, these programming tools offer different degrees of learning ability or collaboration [9]. Myers [13], when presenting his taxonomies of visual programming and program visualization in 1990, argued that textual languages seem to be more appropriate for professionals. On the other hand, there is a great interest in systems which use graphics in the programming, debugging, and understanding of computer science concepts. Additionally, there are several visual languages which could support the teaching process. Users of all ages can use them in order to create their own code [13]. In a different approach, Webb is suggesting a number of platforms which successfully could support educational applications [20].

Nevertheless, learning to program is usually very hard. Novice programmers come to deal with many difficulties and challenges. Educators should provide students with an effective learning environment. The appropriate knowledge to be taught and the teaching strategies to be used need to be carefully taken into consideration [17]. From the same perspective, Saeli has underlined the importance of computer programming learning as it could enhance students' problem solving skills especially in a multi-disciplinary environment. He also suggests the careful selection and design of the programming knowledge content, due to the difficulties that students encounter when learning programming [19].

Computer programming skills could also enhance computational thinking (CT). According to Wing, CT involves concepts like "problem solving, designing systems, and understanding human behavior" [21]. Under this perspective, CT could be considered as an attitude and skill for everyone and not just the computer scientists. Grover and Rea deal with the concept and explore appropriate tools, environments, assessment methods, and strategies about computational thinking (CT) development in K-12 education [5].

Nevertheless, the various tools, the possible benefits, and the challenges of computing lessons implementation in K-12 CSE still need more exploration. The multiple needs and expectations of the students in the digital era make this need more demanding. Successful approaches could help scholars and educators to provide their students with more learning opportunities and useful skills.

III. METHODOLOGY

A. Overview

For the context of this study, we have considered a peer reviewed search in the following international online bibliographic databases: AACE Digital Library, Academic Search Premier, Association for Computing Machinery Digital Library, and EBSCO Education Source included ERIC. Our searching key terms were "learning" and "programming" and "young students". We also included the terms "games", "constructionism" or "do it yourself". The examined period was five years, from January 2009 to December 2013. This process was conducted independently by two experts, a CS educ. researcher and a research librarian and resulted in 1514 articles.

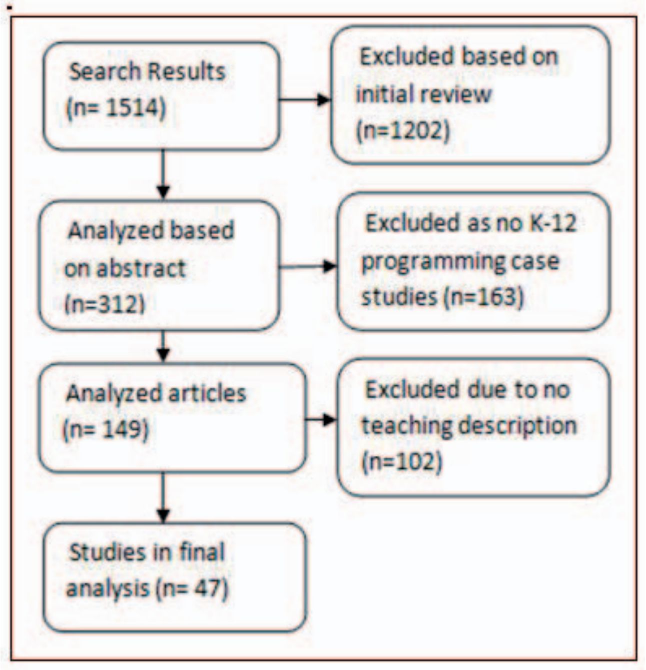
Search Results (n= 1514)
Excluded based on initial review (n=1202)
Analyzed based on abstract (n=312)
Excluded as no K-12 programming case studies (n=163)
Analyzed articles (n= 149)
Excluded due to no teaching description (n=102)
Studies in final analysis (n= 47)

Figure 1: Papers Selection Methodology

A first step of this effort was to exclude papers describing case studies in process, posters, conference reports, and not related papers. Then, based on the papers' abstracts, we excluded articles irrelevant to programming curricula and tools and not K-12 students learning. Afterwards, we were based on the whole content of each article in order to explore the various technology based pedagogical approaches used in K-12 CSE. We found several papers which were concerning programming tools (e.g. Alice [69], Scratch [52], Kodu [63]), virtual worlds programming tools (e.g. SLurtles [4]), and programming video games (e.g. Game Gidget [11], the Machineers [12] or the Smart Lady Game [22]). Additionally, many papers were referring not only to the typical school environment but also to camps, after school or summer programs, like the Beowulf

Bootcamp [1], the Digital Divas [10], and many others. In this step, we decided to exclude papers without a detailed description of the teaching design process or with informal learning as it would be impossible for us to validate the teaching process integration. Finally, we examined the content of the remaining papers with focus on the research methodology, and the quality of the results evaluation. Finally, we decided to include some surveys which didn't describe an instructional method, due to the lack of papers referring to tangible programming interfaces. We finally chose a total number of forty seven papers for the needs of this review. A list of the selected papers can be found at the end of the reference section of the paper. We believe that the included articles can provide us with a useful guideline of teaching approaches aiming at teaching computing in K-12 education.

B. Types of Papers collected

An analysis of these 47 studies context revealed our classification scheme, which is described by the following categories:

Programming tools

Many tools have been introduced in order to teach computing to young students in K-12 CSE. Most of them are modern, offering a usable interface with many opportunities, and aiming at making computing more attractive. In categorizing the various programming tools, we have based on the following assumptions. A text programming language may include a text or a graphical user interface. The programming process though means typing text lines. When a programming language includes primarily visual expressions, then it is a visual (graphical) one. This graphical interface is more usable to the novice learners. The visual programming process needs the ability to map various on-screen symbolic representations like icons and other graphical objects to their results [2]. Finally, in the tangible interfaces, learners have the opportunity to manipulate directly tangible objects which actually form the generated code [56].

The various programming languages which have been used for educational purposes could be generally categorized into textual, visual, and tangible according to the "look" of their code. In our research, we distinguished approaches using a text based (n=12), a visual (n=29), or a tangible programming language (n=3). Finally, in 3 papers more than one programming tool were employed.

Educational context

Most of these research papers present: various computing concepts (e.g. control flow and variables), skills (e.g. procedural thinking), and practices (e.g. debugging) in a project based methodology. Our focus though, was not on the programming curriculum itself, but mainly on the educational context, which has been used in order to motivate and enhance students learning. Under this perspective, we distinguished the following categories: game design and development (n=17), programmable physical - tangible tools (n=18), modeling and computation (n=3), and music generation (n=1). There were also (n=8) papers describing more than one or not exactly defined contexts.

A summary of the benefits and disadvantages of the most popular approaches in computing education indicates different options and challenges. Although we need to highlight that it is not easy to decide the most suitable approach, as long as each case has different characteristics (e.g. students' backgrounds, experience, and expectations).

Instructional methods

Teaching programming needs further abilities than knowing how to program. Several computing concepts must be represented and formulated in order to make comprehension effective. In addition, many difficulties have to be overcome [19]. In our review work, we have focused on the instructional methods used with the various programming tools and educational contexts, with the aim to motivate and enhance students' learning. Under this perspective, categorization is not an easy issue. In our approach, we distinguished various methods while teaching such as text, visual, tangible, and game programming, modeling, and programmable objects - platforms.

We do not specify methods for the K-12 CSE, but we only provide with useful guidelines aiming at helping educators' decisions and posing/leading/guiding questions for further research.

IV. RESULTS AND DISCUSSION

Many researchers indicate the importance of a creative, technology-enhanced learning in order to broaden CSE [34][45] and to provide young students with skills like computational thinking [21] and creativity [15][16]. Under this perspective, introducing and enhancing computing to young students in an enjoyable way, is not something new but remains an interesting and demanding issue. In our research, several papers concerning the learning of computing concepts, principles and practices have been introduced. Additionally, this concern seems to grow over the last years (figure 2).

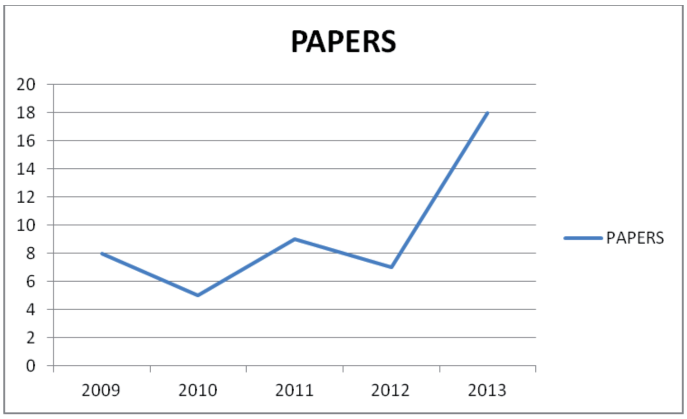

Figure 2: Number of Papers published during the last five years (2009 - 2013)

In particular, our study performs an exploration of programming tools, instructional methods, and educational contexts, in K12 field.

A. *Programming tools in K-12 education*

Various programming tools have been used in computing lessons for young students. In this literature 33 different programming tools have been listed (see Appendix). Most of them are modern, offering usable interface with many affordances and aim at positing programming as an attractive, creative and easy for all skill.

According to our study, most researchers have decided to use visual programming languages (e.g. Modkit [54], Scratch [52] – S4A [32] or App Inventor [35]) in introductory programming lessons. There are though certain approaches which choose to use an "authentic" and powerful text based programming environment (e.g. Action Script [53], Arduino Integrated Development Environments (IDEs)) [34][37] [54]and in some cases a tangible one (e.g. Robo-Blocks[61], Tern[38]). As a summary, it is important to highlight that there are many and diverse features to be considered when deciding the appropriate programming tool.

Textual programming and professional tools can support the instruction of computing concepts [40]. They form an authentic technology [53] which can offer a pleasant learning experience [49], which in turn motivates students to engage in programming [41]. Many times, learning programming could be challenging for young students [53], and instruction programming has also certain challenges for teachers even with well-chosen materials [41]. On the other hand, some students might be more inspired by authentic and professional tools [49].

Visual programming language provide young students with an engaging introduction to computing [23] and successfully support the instruction of important CS concepts [52]. Additionally, skills like algorithmic thinking, managing faulty situations [39] and digital fluency [39][67] could be supported. At this point, we should mention that visual programming languages include features which provide accessibility to students with disabilities. In particular, students with learning or behavioral disabilities who are able to use devises like mouse, find this type of environments deeply absorbing [23]. When using a visual programming language, the computing concepts should be carefully explained and pedagogically supported [37]; in addition, the appropriate support should be provided for the case of complex aspects and notions [29]. A negative parameter could be the extra time devoted by teachers, in order to learn the technical and pedagogical aspects of recently developed programming environments [52].

A tangible interface could promote collaborative interaction [38] and also make computing enjoyable and attractive by encouraging children- especially the girls- in active exploration and learning [38][56]. Younger children find this interface easier to use [56] and engaging [61], however older children do not seem to consider it as the easiest one. Nevertheless, they still enjoy the experience [56].

Finally, we should also report that often both, learners and educators, worry about and experiment with the transitioning from an easier interface to a more difficult and powerful one in order to be engaged in computing activities. An interesting effort [54] validated the effectiveness of combining Modkit which is a visual programming tool with the LilyPadProtoSnap board to introduce computing to young students. They additionally confirmed that students can successfully transit from building programs in Modkit to writing C code. In the first approach, students programmed while using Modkit and then replicated and added functionality by programming in C using the Arduino development environment. In another approach, students built one program using Modkit, and then repeated the activity, using exclusively the Arduino development environment [54].

Deciding for the most appropriate programming tool is a hard and sometimes complex decision, in most of the cases it depends on its effectiveness to some specific learning goals and also its adequacy for students' needs and background.

B. *(Different) Educational contexts of Computing in K-12*

When we select the educational contexts, which will be used in order to motivate and enhance computing learning, lots of parameters should be taken into account. The learning activities should be attractive to the students. Under a constructivism / constructionism perspective, constructing tangible artifacts facilitate students to build their knowledge [14]. Recently, developed programming tools combined with attractive educational contexts can support such approaches. Knowing the various options and challenges of several context types could be useful to educators.

Game design and development is a very popular approach especially in introducing computing to students. There are several tools like Scratch [28], Greenfoot [60], and Action Script [53] which can motivate young students in game design and development. Additionally, game modification "modding" [50] or code remixing [28] could be used instead, especially due to certain "class-workshop"-time constraints. Many researchers argue that game development is an enjoyable learning experience [24][27][53], which supports a profound learning of computer science concepts [24][29][53], higher-order thinking, and abstraction skills [27] and also provides self-confidence [24]. It is important to mention here that due to limitations like luck of time [65], teacher's competency in game development curricula and not just programming [24], and managing student expectations [31] game development is not always possible.

Tangible construction kits and robotics are very popular approaches in K-12 computing education. There are various programmable hardware platforms which offer an authentic and stimulating learning experience like Arduino [32][33][34][37], Lilypad Arduino [42][44][45][54], .NET Gadgeteer hardware [59], Lego Mindstorms [47][56][63], Makey – Makey [28], Pico Cricket board [62], or tangible mobile computing [35][64]. The tangible nature of such physical tools offers a positive [34], exciting, and productive learning experience [28][45][59] which draws a diverse population [54]. Electronic components (controllers) and other prototyping kits and tools can increase students' comfort, enjoyment, and interest with programming [37][45][47][54]. Problem-solving, expressive hands-on making and other constructionism practices could also be supported [44]. Employing tangible or embodied interfaces seems to promote computing concepts and practices [34][42][45] , contribute to broadening participation in CSE [42][45] and engineering in general [51]. We should also mention that due to the common use of assistive technology and open source software; robotics has offered extra affordances to visually impaired students and have achieved to increase their interest and confidence [47]. Mobile computing also provides a powerful new context for motivating computational thinking [35].

Modeling and simulation aims at teaching primarily scientific concepts using computer programming [25][57][58]. Students consider computer modeling as an attractive way of learning programming [25]. For example, deep, conceptual understandings can be achieved through activities which integrate modeling, programming and physics [57][58]. On the other hand, programming itself can be difficult to learn. Integrating programming with learning other science concepts can introduce challenges for students that pertain to learning programming but may not be relevant for understanding scientific concepts [58]. It is necessary that both teachers and students have some fluency with the used programming language in addition to the relevant domain knowledge [57].

Finally there are several approaches in which computing concepts are taught through various activities based on the appropriate each time curriculum [52], scaffold examples [66], challenging applications [49], and music [48]. It is also critical that, educators won't give the impression that CS deals with trivial matters [49].

Deciding the most appropriate contexts depends on many parameters like the various programming tools features, students' age and experience, teaching goals etc. It is not an easy decision, especial when students have different interests, backgrounds, experience, and expectations.

C. Instructional methods in K-12 computing education

When teaching computing, lots of parameters need to be considered, like the students' age and experience or the learning goals. Alternative methodologies provide educators with the opportunity to deal with the complexity of each formal or informal class. A popular methodology widely used in computing education is a problem – project based approach, when a student comes to implement his knowledge by solving a problem or creating a project. Following a step by step instructional procedure, could help learners to successfully create their project especially within the limited time of the school environment. This option though, is a negative factor for the students' creativity [26]. Educators very often come to deal with this dilemma. Alternative creative instructional approaches which support young students' active involvement in the learning process, according to the constructivism and constructionism principles, could be very helpful. Scalable game design or the "use – modify – create" approach are such examples. Educators could additionally use discussions, presentations, demonstrations, handouts, and tutorials to motivate young students and also enhance learning.

There are several methods used by educators and researchers in order to deal with the complexity and the needs of various cases. The use of them though could be adapted in more approaches with similar features:

For example, a text based programming language could offer an authentic experience but it is also a challenging approach. In most cases, students come to create their own code, using some help by instructors or peers [40][48]. A common practice which simplifies the procedure is to give a "piece" of code to the students asking them to study, modify and extend it. This could be helpful and also effective especially for complex examples or shortage of time [41][49]. Additionally, programming activities combined with kinesthetic ones could also provide a better understanding of quite complex programming concepts [36].

A common practice when teaching game development follows three steps: first, students play games in order to develop some experience about games. Then, they practice their programming skills using examples or challenges and finally they create their own video game [24][60][67][68][69]. This approach is widely known as Use – Modify – Create. Similarly, there is another program (Scalable Game Design) which is running in USA. This one fosters creativity in the public educational structure as by scaffolding, students' progress from simple arcade games to more sophisticated ones [26].

Many educators underline the effectiveness of the "in time"pedagogy which follows a non sequential presentation of the various concepts. In particular, new knowledge is provided whenever necessary, through various activities and under a project – based approach. Of course the projects must be carefully selected depending on the different each time learning goals [52].

Common strategies when students learn to use tangible / physical objects while programming are the various demonstrations / instructions or tutorials followed by creative sessions in which help is provided when it is asked. In many of these cases, projects' presentations provide a good motivation to the learners [32][34].

Programming could also be considered as a skill that could facilitate learning in other domains. From this apprenticeship perspective, teachers support students in achieving their goals due to the complexity of various concepts [6], [7]. Support and scaffolding provide students with a better understanding of various difficult computational [46][66] and science [57][58] concepts and practices. In addition, Inquiry Based Learning is a powerful learning model where the instructor presents the problem and through questions helps the students to solve it. This way, the instructor and the students complete the project together [7][69].

In a tangible programming approach, the instructional method might includes several debugging tools combined with a floor robot aiming at understanding or extending the code [61].

A collaborative environment could be also a very important aspect of designing a learning process. The various advantages of a collaborative environment could support learning. Higher achievements, positive relationships among students as well as a healthier psychological adjustment could be such examples [8]. The most common practice in computing learning is the adoption of pair programming where two students write the same code simultaneously in order to solve / create the same problem / project. There is evidence though, that there is some positive influence on friend partnerships but not on the non-friend ones [69].

Finally, the successful transitioning between frameworks could be effective when using a visual or tangible programming language. It is possible that some students will need to extend their skills using more powerful and professional tools. Some researchers have attempted, by using an “easier” interface, to teach an activity and then repeated the same activity by using a more complicated one, as it is described above. These kinds of transitioning assist students to move to more complex and maybe more powerful implementations and also to develop a deeper understanding of the various programming concepts [54].

It is clear that there are a lot of alternatives when an educator comes to deal with several difficulties and limitations. Knowing the advantages and disadvantages could be very helpful in making decisions and helping students. Unfortunately, there are no rules to be followed only decisions to be made. This challenge is the obstacle and also the charm of being a teacher anyway.

V. CONCLUSIONS

After reviewing over 47 studies on computing, there is a common sense that implementing computing lessons in K-12 education could be an enjoyable and effective learning experience. A successful implementation though needs many decisions to be made. Choosing between the various programming tools, activities, and instructional methods is a quite challenging issue. Different needs and expectations could lead us to different choices. It could be quite helpful to have some guidelines from similar successful or not approaches.

There are various perspectives when deciding the most appropriate programming tool to be used in a computing activity. Textual programming and professional tools are an authentic technology which can support learning [53][65]. There is no doubt though that for many students text based languages could be very hard. These disadvantages could be overwhelmed by the feeling that students are engaged with a professional tool in order to create an authentic project. This option could be interesting in attracting students with more expectations and needs (RQ1). On the other hand, a visual programming language provides young people a positive introduction to computing and can support the successful learning of important CS concepts and skills [23][52]. The advantages of an option like this are also great as visual tools make computing accessible to more people. In the typical school environment for example, many students could be benefit by a visual programming experience. Additionally, educators have the option of transitioning their students from one interface to the other. This could be useful in some class settings (RQ1). From the same perspective, a tangible interface supports programming in a collaborative environment and could make programming attractive to the younger children [38][56].

There are several interesting educational contexts which could successfully motivate K-12 computing education. Game development for example could be an enjoyable learning experience which supports deeply learning of computer science concepts [24][29][53]. The tangible nature of physical tools could also offer a positive learning experience which also promotes computing concepts and practices [34][42][45]. In addition, modeling and programming aim at deep, conceptual understandings [57][58]. Literature overview revealed that educational contexts in K-12 computing education should be carefully considered for motivating students and improving their learning (RQ2).

Concluding, the programming tools and activities which could support specific learning goals is a hard and crucial decision. The appropriate instructional method though is a real challenge. It is difficult to find a class with the same features with another as learners can be different in many ways. That makes conclusions impossible. Only alternative methodologies could provide educators the opportunity to deal with the complexity of each formal or informal class and support different students needs. Creative sessions where students create / modify projects or solve problems should be a significant part of the learning process. We believe that encouraging students in creative sessions with much of scaffolding is a good starting point (RQ3).

A number of suggestions for further research have emerged from reviewing prior and ongoing work on K-12 computing education. Recommendations for future research could be the features of various programming tools which could support computer science concepts, skills, and practices deep learning in an enjoyable way and also different students’ needs and expectations. Additionally, exploring the instructional methods used in the transitioning between textual, visual, and tangible programming tools could be another interesting aspect.

VI. APPENDIX

Table 1: Review’s Programming Tools

Programming tool	Description
Action Script (http://www.adobe.com/devnet/actionscript.html)	Action Script is an object-oriented programming language for the Adobe Flash Player and Adobe AIR runtime environments.
Agent Cubes (http://www.agentsheets.com)	Agent Cubes is a 3D game design & programming tool, using a visual but not drag and drop interface.
AgentSheets (http://www.agentsheets.com)	Agent Sheets is a programming tool used to create games and simulations through a rule-based, drag-and-drop interface.
AIA (http://appinventor.mit.edu/explore/about-us.html)	MIT App Inventor is a blocks-based programming tool to build apps for Android devices
Alice (http://www.alice.org)	Alice is a 3D oriented programming language to create animations through a drag and drop graphical interface.
Arduino IDE (http://arduino.cc/en/guide/Environment)	The Arduino development environment is used to upload programs to the Arduino hardware and communicate with them. Its interface also includes a text editor for writing code.
Bricklayer	Bricklayer is a text-enhanced graphical-based programming environment where students use a drag and-drop interface which frees them from the syntax worries

	while at the same time presents them the syntactic statements (Lau, Winnie WY, et al., 2009).
BricxCC (http://bricxcc.sourceforge.net)	Bricx is an integrated development environment (IDE) for programming LEGO MINDSTORMS robots
C# (Gadgeteer Environment) (http://www.netmf.com/gadgeteer)	.NET Gadgeteer is a platform for creating electronic devices using hardware modules and also provides a powerful programming environment (C#).
CTSiM (http://www.teachableagents.org/research/simulation.php)	CTSiM is a combination of visual programming, modeling, and simulation using an easy to use drag and drop interface.
Delphi (http://www.delphibasics.co.uk/)	Delphi is an text based object oriented Language
DevkitPro (http://devkitpro.org/)	Homebrew development tool
F# (http://fsharp.org/)	F# is a functional-first programming language for complex computing problems with simple, maintainable and robust code.
Game Maker (https://www.yoyogames.com/studio)	Game Maker is an IDE for cross-platform games development without prior programming knowledge. In addition though, game control is supported by a scripting language (GML)
Greenfoot (http://www.greenfoot.org/door)	Greenfoot is a visual and interactive programming environment with a usable interface for object orientation using Java.
Java Bridge (App inventor) (Android SDK tools)	The App Inventor Java Bridge helps on the transition from developing Android apps with AIA, to developing apps with Java and the Android SDK.
Jypeli (C#)	Jypeli is an open-source game programming library
Kodu (www.kodugamelab.com)	Kodu is a simple visual programming language to create games for the PC or the Xbox.
Lego NXT-G (http://www.legoengineering.com/program/nxt-g/)	NXT-G is a programming language for the LEGO NXT using a drag-and-drop, graphical interface.
Logo Programming Language (http://el.media.mit.edu/logo-foundation/index.html)	The Logo Programming Language is a learning tool which support interactivity, modularity, extensibility, and flexibility of data types
Microworlds EX (http://www.microworlds.com/solutions/mwex.html)	MicroWorlds EX is a set of tools to create science simulations, mathematical explorations, and interactive multimedia stories
Modkit (http://www.modkit.com/about)	Modkit is an environment for programming and engineering in a visual and browser-based interface
PicoBlocks Software (http://www.picocricket.com/download.html)	PicoBlocks software supports an easy to use, graphical blocks based interaface.
Python (Earsketch) (http://earsketch.gatech.edu)	EarSketch's software toolset enables students to create music by manipulating loops, composing beats, and applying effects with Python code.
Robo-Blocks	The Robo-Blocks system controls the movement of a floor robot by physical command blocks which can be snapped together through magnetic connectors. The chain of command blocks are then attached to a master block which interprets the program sequence and transmits the commands wirelessly to the floor robot.
Scratch - S4A (http://scratch.mit.edu) (http://s4a.cat)	-Scratch is a programming language for games, and animation creation using a graphical, drag and drop interface. Scratch is designed and maintained by the Lifelong Kindergarten group at the MIT Media Lab. -S4A is a Scratch modification for the Arduino hardware platform programming using new blocks for managing sensors and actuators connected to Arduino.
Script Ease (Aurora Toolset)	Aurora Toolset is a game construction toolset to specify interactions through the use of patterns. The most authors never see the scripting code which is generating automatically by ScriptEase
Snap (http://snap.berkeley.edu/)	Snap (BYOB) is a visual, drag-and-drop programming language that allows building blocks. It includes concepts suitable for a serious introduction to CS for high school or college students.
Stagecast Creator (http://www.stagecast.com)	Stagecast Creator is an easy-to-learn, easy-to-use software tool for making games and simulations. The rules of behavior are generating by moving the characters around (by demonstration).
T_ProRob	The T_ProRob (tangible) system consists of 28 commands and 16 smaller parameters, all cubic shaped. Users connect the cubic commands and parameters and the program's execution starts by pressing a button on the top of the basis (''master box'').
Tern (http://hci.cs.tufts.edu/tern/)	Tern is a tangible programming language to create physical computer programs using interlocking wooden blocks. The shape of the interlocking blocks creates physical syntax (no invalid programs) and the tern programs can be compiled by the pressing of a button.
ViMAP (http://www.visualprogramming.org/p/about-this-site_29.html)	ViMAP is a multi-agent based visual programming language to support science learning through scientific modeling and computational thinking.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the research librarian, Christopher Lawton.

REFERENCES

[1] S. R. Brandt, C. Dekate, P. LeBlanc, & T. Sterling, "Beowulf bootcamp: teaching local high schools about HPC," In Proceedings of the 2010 TeraGrid Conference, p. 4, ACM, August 2010.

[2] M. M. Burnett, "Visual programming," Wiley Encyclopedia of Electrical and Electronics Engineering, 1999.

[3] Y. Eshet, " Digital literacy: A conceptual framework for survival skills in the digital era," *Journal of Educational Multimedia and Hypermedia*, Vol. *13*(1), pp. 93-106, 2004.

[4] C. Girvan, B. Tangney, & T. Savage, "SLurtles: Supporting constructionist learning in< i> Second Life</i>," Computers & Education, Vol. 61, pp. 115-132, 2013.

[5] S. Grover, and R. Pea, "Computational thinking in K–12. A review of the state of the field," Educational Researcher, Vol. 42.1, pp. 38-43, 2013.

[6] M. Guzdial, "Software-realized scaffolding to facilitate programming for science learning," Interactive Learning Environments, Vol. 4.1, 001-044, 1994.

[7] C. E. Hmelo-Silver, R. G. Duncan, and C. A. Chinn, "Scaffolding and achievement in problem-based and inquiry learning: A response to Kirschner, Sweller, and Clark (2006)," Educational Psychologist, Vol. 42.2,pp. 99-107, 2007.

[8] D. W. Johnson, R. T. Johnson, &K. A. Smith, " Cooperative learning: Increasingcollege faculty instructional productivity", ASHE-ERIC Higher Education Rep. No. 4, 1991.

[9] C. Kelleher, R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys (CSUR), Vol.* 37.2 pp. 83-137, 2005.

[10] C. Lang, A. Craig, J. Fisher, & H. Forgasz, "Creating digital divas: scaffolding perception change through secondary school and university alliances," In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, pp. 38-42, ACM, June2010.

[11] M. J. Lee, A. J. Ko, & I. Kwan, "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, August 2013.

[12] H. Lode, G. E. Franchi, & N. G. Frederiksen, "Machineers: playfully introducing programming to children," In CHI'13 Extended Abstracts on Human Factors in Computing Systems, pp. 2639-2642, ACM, April 2013).

[13] B. A. Myers, "Taxonomies of visual programming and program visualization,"Journal of Visual Languages & Computing, Vol. 1.1 pp. 97-123, 1990.

[14] S. Papert,and I. Harel, "Situating constructionism," *Constructionism*36: 1-11. 1991.

[15] S. Papert, and M. Resnick, "Technological Fluency and the Representationof Knowledge," Proposal to the National Science Foundation, MIT MediaLaboratory, 1995.

[16] M. Resnick, "Rethinking learning in the digital age," 2002.

[17] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion,"*Computer Science Education* Vol. 13.2, pp.137-172, 2003.

[18] J. M. Roschelle, R. D. Pea,C.M. Hoadley, D.N. Gordin, & B. M. Means, "Changing how and what children learn in school with computer-based technologies", . The future of children, pp. 76-101, 2000.

[19] M. Saeli, J. Perrenet, W. M. Jochems, &B. Zwaneveld, "Teaching programming in secondary school: a pedagogical content knowledge perspective," Informatics in Education-An International Journal, vol 10_1, pp. 73-88, 2011.

[20] H. Webb, "Computer applications for the classroom: A review," Journal of Computing Sciences in Colleges, Vol. 27(3), pp. 65-72, 2012.

[21] J. M. Wing, "Computational thinking," Communications *of the ACM*, Vol. *49*(3), pp. 33-35, 2006.

[22] A. Zeid, G. Al-Mirza, & L. Al-Meshwah, "SLG: an online educational simulation game to teach programming concepts," In Proceedings of the Second Kuwait Conference on e-Services and e-Systems, p. 15, ACM, April 2011.

SYSTEMATIC REVIEW REFERENCES

[23] J. C. Adams, "Scratching middle schoolers' creative itch", In Proceedings of the 41st ACM technical symposium on Computer science education, pp. 356-360, ACM, March 2010

[24] M. Al-Bow, D. Austin, J. Edgington, R. Fajardo, J. Fishburn, C. Lara, & S. Meyer, "Using game creation for teaching computer programming to high school students and teachers", ACM SIGCSE Bulletin, Vol. 41(3), pp. 104-108, 2009.

[25] J. Benacka, &J. Reichel, "Computer Modeling with Delphi,"InInformatics in Schools. Sustainable Informatics Education for Pupils of all Ages, pp. 138-146, Springer Berlin Heidelberg, 2013.

[26] V. E. Bennett, K. H. Koh, &A. Repenning, "A. CS education re-kindles creativity in public schools," In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, pp. 183-187, ACM, June 2011.

[27] M. Carbonaro, D. Szafron, M. Cutumisu, &J. Schaeffer, "Computer-game construction: A gender-neutral attractor to Computing Science," Computers & Education, Vol. 55(3), pp. 1098-1111, 2010.

[28] R. Davis, Y. Kafai, V. Vasudevan, &E. Lee, "The education arcade: crafting, remixing, and playing with controllers for scratch games," InProceedings of the 12th International Conference on Interaction Design and Children, pp. 439-442, ACM, June 2013.

[29] J. Denner, L. Werner, &E. Ortiz, "Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts?" Computers & Education, Vol. 58(1), pp. 240-249, 2012.

[30] J. P. Dimond, S. Yardi, &M. Guzdial, "Mediating programming through chat for the OLPC," In CHI'09 Extended Abstracts on Human Factors in Computing Systems, pp. 4465-4470, ACM, April 2009.

[31] K. Doran, A. Boyce, S. Finkelstein, &T. Barnes, "Outreach for improved student performance: a game design and development curriculum," InProceedings of the 17th ACM annual conference on Innovation and technology in computer science education, pp. 209-214, ACM, July 2012.

[32] M. N. Giannakos, &L. Jaccheri, "Designing creative activities for children: the importance of collaboration and the threat of losing control," InProceedings of the 12th International Conference on Interaction Design and Children, pp. 336-339, ACM, June 2013.

[33] M. N. Giannakos, &L. Jaccheri, "What motivates children to become creators of digital enriched artifacts?" In Proceedings of the 9th ACM Conference on Creativity & Cognition, pp. 104-113, ACM, June 2013.

[34] M. N. Giannakos, L. Jaccheri, &R. Proto, "Teaching Computer Science to Young Children through Creativity: Lessons Learned from the Case of Norway," In Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research, pp. 103-111, Open Universiteit, Heerlen. April 2013.

[35] S. Grover, &R. Pea, "Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students," In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 723-728, ACM, March 2013.

[36] K. Gunion, T. Milford, &U. Stege, "Curing recursion aversion," ACM SIGCSE Bulletin, Vol. 41(3), pp.124-128, 2009.

[37] M. Høiseth, &L. Jaccheri, "Art and technology for young creators," InEntertainment Computing–ICEC, pp. 210-221, Springer Berlin Heidelberg, 2011.

[38] M. S. Horn, E. T. Solovey, R. J. Crouser, &R.J. Jacob, "Comparing the use of tangible and graphical programming languages for informal science education," In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 975-984, ACM, April 2009.

[39] A. Ioannidou, A. Repenning, &D. C. Webb, "AgentCubes: Incremental 3D end-user development," Journal of Visual Languages & Computing, Vol. 20(4), pp. 236-251, 2009.

[40] V. Isomöttönen, A. J. Lakanen, &V. Lappalainen, "K-12 game programming course concept using textual programming," In Proceedings of the 42nd ACM technical symposium on Computer science education, pp. 459-464, ACM, March 2011.

[41] G. Kacmarcik, &S. G. Kacmarcik, "Introducing computer programming via gameboy advance homebrew," . In ACM SIGCSE Bulletin, Vol. 41, No. 1, pp. 281-285, ACM, March 2009.

[42] Y. B. Kafai, K. Searle, E. Kaplan, D. Fields, E. Lee, &D. Lui, "Cupcake cushions, scooby doo shirts, and soft boomboxes: e-textiles in high school to promote computational concepts, practices, and perceptions," In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 311-316, ACM, March 2013.

[43] R. Khaled, "Equality= inequality: probing equality-centric design and development methodologies," In Human-Computer Interaction–INTERACT, pp. 405-421, Springer Berlin Heidelberg, 2011.

[44] S. Kuznetsov, L. C.Trutoiu, C. Kute, I. Howley, E. Paulos, &D. Siewiorek, "Breaking boundaries: strategies for mentoring through textile computing workshops," In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2957-2966, ACM, May 2011.

[45] W. W. Lau, G. Ngai, S. C. Chan, &J. C. Cheung, "Learning programming through fashion and design: a pilot summer course in

wearable computing for middle school students," ACM SIGCSE Bulletin, Vol. 41(1), pp. 504-508, 2009.

[46] C. M. Lewis, "Is pair programming more effective than other forms of collaboration for young students?" Computer Science Education, Vol. 21(2), pp. 105-134, 2011.

[47] S. Ludi, &T. Reichlmayr, "The use of robotics to promote computing to pre-college students with visual impairments," ACM Transactions on Computing Education (TOCE), Vol. 11(3), pp. 20, 2011.

[48] B. Magerko, J. Freeman, T. McKlin, S. McCoid, T. Jenkins, &E. Livingston, "Tackling engagement in computing with computational music remixing," In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 657-662, ACM, March 2013.

[49] G. Maggiore, A. Torsello, &F. Sartoretto, "Engaging high school students in computer science via challenging applications," InProceedings of the 2011 conference on Information technology education, pp. 43-48, ACM, October 2011.

[50] D. Makris, K. Euaggelopoulos, K. Chorianopoulos, &M. N. Giannakos, "Could you help me to change the variables?: comparing instruction to encouragement for teaching programming," In Proceedings of the 8th Workshop in Primary and Secondary Computing Education, pp. 79-82, ACM, Novenber 2013.

[51] G. Marcu, S. J. Kaufman, J. K. Lee, R. W. Black, P. Dourish, G. R. Hayes, &D. J. Richardson, "Design and evaluation of a computer science and engineering course for middle school girls," In Proceedings of the 41st ACM technical symposium on Computer science education, pp. 234-238, ACM, March 2011.

[52] O. Meerbaum-Salant, M. Armoni, &M. Ben-Ari, "Learning computer science concepts with scratch," Computer Science Education, Vol. 23(3), pp. 239-264, 2013.

[53] C. C. Navarrete, "Creative thinking in digital game design and development: A case study," Computers & Education, Vol. 69, pp. 320-331, 2013.

[54] K. Qiu, L. Buechley, E. Baafi, & W. Dubow, "A curriculum for teaching computer science through computational textiles", In Proceedings of the 12th International Conference on Interaction Design and Children, pp. 20-27, ACM, June 2013

[55] M. Sands, J. Evans, &G. D. Blank, "Widening the K-12 pipeline at a critical juncture with Flash™," Journal of Computing Sciences in Colleges, Vol. 25(6), pp. 181-190, 2010.

[56] T. Sapounidis, &S. Demetriadis, "Tangible versus graphical user interfaces for robot programming: exploring cross-age children's preferences," Personal and ubiquitous computing, Vol. 17(8), pp. 1775-1786, 2013.

[57] P. Sengupta, &A. V. Farris, "Learning kinematics in elementary grades using agent-based computational modeling: a visual programming-based approach," In Proceedings of the 11th International Conference on Interaction Design and Children, pp. 78-87, ACM, June 2012.

[58] P. Sengupta, J. S. Kinnebrew, S. Basu, G. Biswas, &D. Clark, "Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework," Education and Information Technologies, Vol. 18(2), pp. 351-380, 2013.

[59] S. Sentance, &S. Schwiderski-Grosche, "Challenge and creativity: using. NET gadgeteer in schools," In Proceedings of the 7th Workshop in Primary and Secondary Computing Education, pp. 90-100, ACM, November 2012.

[60] S. Simmons, B. DiSalvo, &M. Guzdial, "Using game development to reveal programming competency," In Proceedings of the International Conference on the Foundations of Digital Games, pp. 89-96, ACM, May 2012.

[61] A. Sipitakiat, &N. Nusen, "Robo-Blocks: designing debugging abilities in a tangible programming system for early primary school children," InProceedings of the 11th International Conference on Interaction Design and Children, pp. 98-105, ACM, June 2012.

[62] B. Tangney, E. Oldham, C. Conneely, S. Barrett, &J. Lawlor, "Pedagogy and processes for a computer programming outreach workshop—The bridge to college model," Education, IEEE Transactions on, Vol. 53(1), pp. 53-60, 2010.

[63] D. S. Touretzky, D. Marghitu, S. Ludi, D. Bernstein, &L. Ni, "Accelerating K-12 computational thinking using scaffolding, staging, and abstraction," In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 609-614, ACM, March 2013.

[64] A. Wagner, J. Gray, J. Corley, &D. Wolber, "Using app inventor in a K-12 summer camp," In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 621-626, ACM, March 2013.

[65] H. C. Webb, &M. B. Rosson, "Exploring careers while learning Alice 3D: a summer camp for middle school girls," In Proceedings of the 42nd ACM technical symposium on Computer science education, pp. 377-382, ACM, March 2011.

[66] H. Webb, &M. B. Rosson, "Using scaffolded examples to teach computational thinking concepts," In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 95-100, ACM, March 2013

[67] L. Werner, J. Denner, M. Bliesner, &P. Rex, "Can middle-schoolers use Storytelling Alice to make games?: results of a pilot study," InProceedings of the 4th International Conference on Foundations of Digital Games, pp. 207-214, ACM, April 2009.

[68] L. Werner, J. Denner, S. Campe, &D. C. Kawamoto, "The fairy performance assessment: Measuring computational thinking in middle school," In Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 215-220, ACM, February 2012.

[69] L. Werner, J. Denner, S. Campe, E. Ortiz, D. DeLay, A. C. Hartl, &B. Laursen, "Pair programming for middle school students: does friendship influence academic outcomes?" In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 421-426, ACM, March 2013.