IJCSES

Volume 2, Issue 2
April 2018
ISSN 2513-8359

International Journal of Computer Science Education In Schools

Editors

Prof. Filiz Kalelioglu

Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

January 2018, Vol 2, No 2

DOI: 10.21585/ijcses.v2i2

Table of Contents

Articles

	Page
Alex Hadwen-Bennett, Sue Sentance, Cecily Morrison Making Programming Accessible to Learners with Visual Impairments: A Literature Review	3 - 13
Ünal Çakiroğlu, Betül Er, Nursel Uğur, Esra Aydoğdu Exploring the Use of Self-Regulation Strategies in Programming with Regard to Learning Styles	14 - 28
Meng-Leong How, Chee-Kit Looi Using Grey-based Mathematical Equations of Decision-making as Teaching Scaffolds: from an Unplugged Computational Thinking Activity to Computer Programming	29 - 46

Making Programming Accessible to Learners with Visual Impairments: A Literature Review

Alex Hadwen-Bennett[1]
Sue Sentance[1]
Cecily Morrison[2]

[1]King's College London
[2]Microsoft Research Cambridge

DOI: 10.21585/ijcses.v2i2.25

Abstract

Programming can be challenging to learn, and for visually impaired (VI) learners, there are numerous additional barriers to the learning process. Many modern programming environments are inaccessible to VI learners, being difficult or impossible to interface with using a screen reader. A review of the literature has identified a number of strategies that have been employed in the quest to make learning to program accessible to VI learners. These can be broadly divided into the following categories; auditory and haptic feedback, making text-based languages (TBLs) accessible, making block-based languages (BBLs) accessible and physical artefacts. A common theme among the literature is the difficulty VI learners have in gaining an understanding of the overall structure of their code. Much of the research carried out in this space to date focuses on the evaluation of interventions aimed at VI high-school and undergraduate students, with limited attention given to the learning processes of VI learners. Additionally, the majority of the research deals with TBLs, this is despite the fact that most introductory programming courses for primary learners use BBLs. Therefore, further research is urgently needed to investigate potential strategies for introducing VI children in primary education to programming and the learning processes involved.

Keywords: visual impairments, programming education, physical programming, special needs

1. Introduction

The introduction of computing into the national curriculum for England in 2014, brought with it the requirement for primary school children to be taught the basic concepts of programming from the age of 5 (Department for Education, 2014). Programming can be challenging to learn and, for visually impaired (VI), learners there are numerous additional barriers to the learning process. Many modern programming environments are inaccessible to VI learners, being challenging or impossible to interface with using a screen reader (Baker et al., 2015; Stefik et al., 2011) and user interfaces often employ highly graphical depictions (Ludi, 2013). Kane & Bigham (2014) identified the following criteria for the development of environments in which VI children can learn to program:

- "Programming tools must be accessible to the student and must work with the assistive technology that he or she uses."
- "The student must be provided with programming tasks that hold their interest and provide encouraging feedback." (Kane & Bigham, 2014, p. 257).

This literature review sets out to provide an overview and discussion of the different strategies that have been employed in order to make learning programming accessible to VI learners. Additionally, areas that require further research will be identified and discussed.

2. Methodology

This review examines literature from peer-reviewed sources, published between 2000 and November 2017. Studies were identified by searching research databases, in addition to citation tracing. The following databases were searched: ACM Digital Library, Taylor and Francis, IEEE, Eric and Wiley Online Library. The search terms "visually impaired", "programming" and "education" were initially used, followed by additional searches employing alternative search terms with similar meanings, an overview of these terms is given in Table 1.

Search Term	Alternatives
Visually Impaired	Blind, visual impairments
Programming	Coding, software development
Education	Learning, learners, school

Table 1 Summary of Search Terms

Once a short list of articles was formed, the following criteria were used to decide whether the articles should be included in the literature review:

- Papers were included if they had an educational focus, however a small number of other papers were retained in order to provide contextual information.
- Papers were included if they were included in a peer-reviewed academic publication.
- Papers published since 2000 were included. One exception was made for a paper that is frequently cited and therefore provides contextual information.

Upon further examination of the literature, four main themes emerged; making text-based languages accessible, making block-based languages accessible, physical artefacts as well as auditory and haptic feedback. Each of these themes is explored in turn in the following sub-sections. An overview of the literature cross-referenced by theme is also provided in appendix A.

3. Overview of Literature

3.1 Making Text-Based Languages Accessible

3.1.1 Accessibility of Programming Environments

A survey of experienced VI developers has demonstrated that many programming environments are either not fully compatible with screen readers or challenging to navigate solely using auditory feedback alone, this makes them inaccessible to many VI programmers (Albusays & Ludi, 2016). For example Eclipse features a number of tabbed windows, which can be accessed through keyboard shortcuts, however this is a time consuming process when relying on auditory feedback (Cheong, 2010). Additionally, the BricxCC and Robot C programming environments, which are both designed for programming Lego Mindstorms robots, are not fully compatible with JAWS (a popular screen reader) (Ludi, 2013). Although Visual Studio (2010) is technically accessible, no sound is generated to indicate when the user switches between tabs (Stefik et al., 2011).

One approach that has been taken to address the inaccessibility of programming environments is the use of a standard text editor alongside a screen reader (Bigham et al., 2008; Cheong, 2010; Kane & Bigham, 2014). A drawback of this approach is the loss of debugging tools that are standard in most modern programming environments. Tools have also been developed to improve the accessibility of programming environments, for example the Wicked Audio Debugger (WAD) was developed to work with the popular Visual Studio programming environment to assist VI programmers with the debugging process (Stefik et al., 2007).

An alternative strategy is the development of accessible programming environments. An example is JavaSpeak, which was developed as a tool to assist VI undergraduate students learn how to program in Java (Francioni & Smith, 2002; Smith et al., 2000). It is based on the concept of EmacSpeak (Raman, 1996), which has a speech interface aimed at experienced programmers. Unlike EmacSpeak, JavaSpeak is designed for undergraduate students that are learning to program, enabling them to experience their code at different granularities. The development process of the JavaSpeak environment has been described, however there is no evidence of evaluation of the tool in use.

More recently, the JBrick programming environment was developed to make the programming of Lego Mindstorms robots accessible (Ludi, 2013). The NXC language (Not eXactly C) has been used in outreach programs along with the BricxCC programming environment to enable VI learners to program Lego Mindstorms robots (Dorsey et al., 2014; Ludi & Reichlmayr, 2011). However, the BricxCC programming environment is not fully compatible with JAWS (a popular screen reader). JBrick was developed as an alternative to BricxCC, it is compatible with common screen readers and braille displays, enables code to be easily located by line number and provides both audio and visual feedback (Ludi et al., 2014).

3.1.2 Accessibility of Programming Languages

Another important consideration is the choice of programming language; many commonly used languages, such as C and Java, make extensive use of non-alphanumeric characters such as brackets and curly braces, which can be challenging to work with using a screen reader. Additionally, the complex syntax of many languages can make typing mistakes more likely and debugging more challenging. Languages such as Ruby, which use mainly text and limit the number of non-alphanumeric symbols are preferable as they are less likely to cause problems with screen readers (Kane & Bigham, 2014). In their study, Kane and Bigham also considered Python, as it meets most of the previously mentioned criteria, however it also uses white space which could be confusing when used with a screen reader. During the course of their study, which took place over a week and involved 12 VI learners, Kane and Bigham found that the students were successful in writing programs in Ruby, however the mispronunciation of some of the terms by the screen reader caused minor challenges.

There are text-based languages that have been designed specifically for VI users, for example the APL (Audio Programming Language) for example, was developed by VI learners for VI learners (Sánchez & Aguayo, 2006). APL features a reduced set of commands which can be accessed and selected through a circular command list, with no requirement to memorise commands. The results of a small usability study of APL indicate that the language enables learners to understand programming concepts and apply them.

In 2011, Stefik et al. conducted an exploratory study to evaluate the accessible programming environment Sodbeans, along with the Hop programming language, which they developed. Sodbeans is aimed at middle and high school students and makes use of audio cues for navigation along with an auditory debugger for the Hop programming language. The findings from the evaluation indicate an increase in learner self-efficacy after participation in a programming workshop that employed Sodbeans and Hop.

The Hop programming language was developed further, becoming Quorum, a language designed for all, while still being accessible to VI learners (Stefik et al., 2011). The development of Quorum was informed by empirical studies investigating the intuitiveness of the syntax of different languages and the accuracy rates of novice programmers using them (Stefik & Siebert, 2013).

3.1.3 Code Navigation

A common theme that occurs among the literature is the difficulty VI learners have navigating their code and understanding the overall structure when using a screen reader (Bigham et al., 2008; Kane & Bigham, 2014; Ludi et al., 2014). This can often result in learners inserting code in the incorrect position. There are steps that can be taken to mitigate these difficulties; in order to gain a better understanding of their position in the code, learners can be encouraged to move the text cursor in order to hear the characters read out. In addition, learners can also be provided with code samples in braille to help them develop an understanding of the overall structure of the code.

The challenge of navigating the code and understanding its structure was considered during the development of StructJumper, a plugin for the Eclipse programming environment which enables VI users to navigate through a program written in Java (Baker et al., 2015). StructJumper generates a tree that is made up of the nested structures contained within the program, this enables the user to easily jump between each nested structure in the code. The participants that took part in a small-scale evaluation of StructJumper found that it helped them speed up their navigation through the code.

3.1.4 Other Considerations

It is also important to consider that the level of vision among VI learners will vary considerably, as will their preferred assistive technologies (Bigham et al., 2008; Ludi et al., 2014). Experience with assistive technologies

may also vary. Bigham et al. (2008) found that students that were already proficient in the use of a screen reader were the most successful. Another factor that can impact on progress of VI learners is their familiarity with keyboard layout, with typing skills also being identified as an important skill for learning to program in a text-based language (Ludi, 2013; Ludi et al., 2014).

Another factor to be considered is the accessibility of tools designed to create graphical user interfaces (GUIs), as existing tools that are employed to generate GUIs are either not accessible or very challenging to use for VI learners. In order to address this issue Siegfried (2006) developed a scripting language to enable VI programmers to produce Visual Basic Forms. More recently, Konecki (2014) developed GUIDL, a tool that enables VI learners to create GUIs for their programming projects. GUIDL was evaluated by a small group of adult novice programmers who found they were able to use the tool to successfully create GUIs that could be used in their own programs.

Although there are a number of studies focusing on teaching VI learners to program in a text-based language (TBL), these mainly focus on high school and undergraduate students. The following section will look at the accessibility of (BBLs), which are targeted at students in primary school.

3.2 Making Block-Based Languages Accessible

When learning how to program a significant amount of time is spent learning the syntax of a specific language; this can potentially hinder the development of an understanding of the core programming concepts. BBLs such as Scratch (Maloney et al., 2010) enable learners to develop programs by snapping blocks together, removing the need for them to learn the complex syntax of a TBL.

BBLs are intrinsically visual and are therefore not accessible to most VI learners. There is a need for an alternative to BBLs such as Scratch (Koushik & Lewis, 2016; Ludi, 2015). One such alternative is Noodle, a programming system for creating sound and music that has program elements which can be inserted and arranged purely using keyboard commands (Lewis, 2014). The concept of Noodle is promising; however, it does not appear to have been trialed with learners and the language used in the audio feedback is not appropriate for primary school children. This makes it an unsuitable choice for the introduction of programming to young VI children.

Ludi (2015) and her team have been working on making the Blockly language accessible to VI learners. The language that Ludi and her team are developing will enable navigation purely by keyboard and also incorporate audio cues in order to communicate the level of nesting. Following on from the work on Noodle, Lewis has been working with Koushik in the development of another accessible Blockly-based language called the Pseudospatial Blocks (PB) language (Koushik & Lewis, 2016). Pseudospatial refers to the distorted nature of the geometry of movement. In PB the learner selects an insertion point using the keyboard and they can select the program element they want from a filtered list; the program elements are filtered by syntactic category. Koushik and Lewis (2016) argue that PB has advantages over visual languages for all learners as invalid program blocks for a given space are filtered out.

The Lady Beetle and World of Sounds programming environments are alternatives to BBLs that were developed in order to introduce young VI children to the basic concepts of programming (Jašková & Kaliaková, 2014). The Lady Beetle programming environment enables the learner to select single word commands, without having to type them. These commands control the movement of a beetle across a grid. As the beetle moves, the coordinates of the current square are read out. World of Sounds, on the other hand, enables learners to create simple programs that produce sequences of sounds.

The development of these accessible BBL alternatives is a promising step forward in the quest to find an accessible alternative to block-based languages, however they could still present learners with difficulties gaining an understanding of the overall structure of their code when using a screen reader. The table shown in appendix A demonstrates that there is still some way to go for BBL research to catch up with TBLs.

3.3 Physical Artefacts

3.3.1 Programmable Devices

The physical nature of programmable devices such as robots make them a common tool for the teaching of introductory programming and it is has been shown to be just as appealing to VI learners (Ludi, 2013). When teaching computing with robotics, the robots can either be pre-assembled or learners can be required to build

their own robots as part of the learning process. This has its own challenges, particularly for VI learners.

Dorsey Rayshun, Chung Hyuk, & Howard (2014) conducted an evaluation of four educational robotics kits during a series of summer workshops, which investigated their suitability for use with VI learners. In each workshop the VI learners were paired with a sighted buddy and tasked with building robots using the various kits. The LEGO Mindstorm RCX was found to be the easiest for VI learners to work with, requiring the least support from their sighted buddies.

A number of studies have been conducted, which investigate outreach programs designed to increase participation of VI students in computing using robotics (Dorsey et al., 2014; Ludi, 2013; Ludi et al., 2014; Ludi & Reichlmayr, 2011). The findings of these studies indicate that after the workshops the confidence level of the students in programming improved, as did their desire to take computing in school or pursue it as a career.

3.3.2 Physical Programming Languages

Most systems used in physical computing, whilst being physical themselves are still programmed using a GUI on a computer. In physical programming languages (PPLs), commands are represented by physical objects which can be joined together to create programs. The Tern PPL uses wooden blocks that can be joined together in order to construct programs. A webcam is used to convert physical into digital code (Horn & Jacob, 2007a, 2007b). Tern was initially evaluated over the period of one week with nine sighted children. The children used Tern to program robots, not all of them were able to understand the effect of their programs on the robot. This may be partially down to the delay between code creation and execution as it has to be converted to digital code using a webcam connected to a computer.

The physical nature of physical programming languages means they have the potential to be a powerful learning tool for VI children, howeverTern itself is not accessible. On the other hand there is Torino, a physical programming language that is designed to be inclusive of VI learners (Thieme et al., 2017). Torino features pods which can be joined together to create programs that produce sound and music. Each pod features dials, which act as parameters and enable the learner to change the sound sample or note and the duration. The physical nature of Torino programs could potentially enable the learner to gain an overall of the structure of the whole program.

3.3.3 3D Models

It is common practice for computing teachers to use diagrams, graphics or animations to illustrate programming concepts such as data structures, “most tools used to teach data structures, algorithmic thinking and basic programming are visually oriented” (Papazafiropulos et al., p. 491). While assistive technologies enable VI learners to access information, they are unable to present a complex concept in a simple form in the same way a visual representation can.

3D models can be used to represent abstract concepts in a way that is accessible to VI learners. As part of their research Stefik et al. (2011) interviewed teachers in one school for VI children and found that where possible new concepts should be introduced through the use of physical objects. In response to this, they developed ‘manipulatives’ for teaching key programming concepts, such as variables. Jašková & Kaliaková (2014) used a tactile table consisting of a 10x10 grid to teach VI children how to write simple algorithms. The children were given the task to write a sequence of commands in a text editor that guided a bee to follow a pre-set path through the tactile grid. The learners would simulate the execution of the program by moving the bee with their hands.

With the advent of 3D printers, 3D models have become much easier to produce. Papazafiropulos et al. (2016) used 3D printed models in a small feasibility study to teach concepts such as data structures and algorithms to VI children. The model they used features cylinders of varying heights, with the height representing the value of the element. The cylinders slot into a tray which represents the array. It was used to teach how sorting and searching algorithms could be applied to arrays.

3D printing was also used by Kane & Bigham (2014) as part of a week-long programming workshop, in which children produced code to generate physical visualizations of data. They found that the ability to generate and print their own tactile maps was extremely engaging for the children, however, the speed of 3D printing was a limitation as they had to be printed overnight. They also identified the need for universal tools that can be used to easily create tactile graphics.

Lego provides a quick and simple method of producing basic 3D models for use in the teaching of programming

concepts to VI learners. Capovilla et al. (2013) discovered this when they employed Lego models in the teaching of sorting and searching algorithms to a small group of adult VI learners. Once the learners had familiarized themselves with the algorithms using the Lego models, they were then asked to solve sorting and searching tasks in a spreadsheet. All participants were able to complete the assigned tasks.

3.4 Auditory and Haptic Feedback

Sounds that vary in tone and pitch can be used to indicate the different states of a physical object or virtual representation, as can haptic feedback in the form of vibrations. PLUMB EXTRA (EXploring data sTRuctures using Audible Algorithm Animation) was developed to enable VI undergraduate students to access simulations of algorithms designed to manipulate data structures (Calder et al., 2007). It is based on PLUMB, a system designed to enable VI learners navigate graphs (Calder et al., 2006). The PLUMB EXTRA system enables learners to explore the state of data structures at any point using a series of audio cues. In the Calder et al. (2007) study, the development of the system is described; however, the evaluation of the system is limited.

During a series of workshops, Dorsey et al. (2014) made use of different piano notes and vibrations in a Wii remote in order to indicate the different states of a robot while navigating a maze. The results are this study indicate that if sufficient haptic and auditory feedback is provided, VI learners are able to perform tasks that are considered to be highly visual.

4. Discussion

This review has demonstrated the dominance of TBLs in the literature, this is despite the fact that in primary computing education BBLs are most prevalent, as highlighted by the recent Royal Society Report (The Royal Society, 2017). According to the national curriculum (Department for Education, 2014), all children in England should learn the basic concepts of programming from the age of 5. However, the inherent inaccessibility of BBLs, along with their widespread use in primary computing lessons can lead to VI learners being excluded from programming lessons. Initial steps have been taken towards making BBLs accessible to VI learners, however there is still a long way to go and more research is needed.

Research relating to the use of TBLs with VI learners has identified the difficulty learners can have in gaining an understanding of the overall structure of their code as can they only listen to one line of code at a time, putting a heavy reliance on short term memory. Even though it has been shown that it is possible to make BBLs accessible to VI learners, this difficulty could still present a barrier for learners. PPLs, on the other hand, could potentially enable VI learners to develop an understanding of the structure of the code through touch, as long as the individual blocks or elements used in the PPL are physically different. Therefore, the use of PPLs with VI learners needs to be investigated in terms of learning processes and possible benefits.

The literature relating to TBLs has identified a number of potential challenges for VI learners in addition to possible strategies to overcome them. This research can be used to inform the teaching of programming to high-school VI learners, however more research is still required. If VI learners are successfully introduced to programming in primary school through PPLs or accessible BBLs, they will enter high-school understanding the basic concepts. This could potentially smooth the transition to TBLs and as a result possibly reduce the significance of some of the challenges currently associated with TBLs. This highlights the urgent need for research into strategies for making programming accessible to primary VI learners.

5. Conclusion

Much of the research carried out in this space to date focuses on the development of interventions and their impact on student perceptions and engagement, with limited attention given to the pedagogy of teaching programming to VI learners. This is certainly an area that warrants further research.

Currently the most popular languages for introductory programming in primary schools in the UK are block-based (The Royal Society, 2017), which are currently not accessible to VI learners. Therefore, there is a need for further investigation into potential accessible alternatives to BBLs, PPLs are a promising candidate given their potential to enable learners to gain an understanding of the overall structure of their code.

6. Summary

A range of studies have investigated ways in which learning text-based languages can be made accessible to VI learners (Bigham et al., 2008; Dorsey et al., 2014; Kane & Bigham, 2014; Ludi, 2013; Ludi et al., 2014; Ludi & Reichlmayr, 2011; Smith et al., 2000; Stefik et al., 2011), however, these have focused mainly on high school and undergraduate students. Block-based languages have also been examined, with the aim of making them accessible to VI learners (Koushik & Lewis, 2016; Lewis, 2014). Pseudospacial Blocks (PB) is a promising development, which is more suited to the needs of VI learners in primary education. It should be noted however, that it could be challenging for learners to gain an understanding of the overall structure of their code when using PB, as is the case with text-based languages.

Physical artefacts can be employed to engage sighted and VI learners alike, the use robotics is one such example (Dorsey et al., 2014; Ludi, 2013; Ludi et al., 2014; Ludi & Reichlmayr, 2011). The drawback of this approach is that it currently still relies on TBLs, bringing with them their own complications, which have based discussed previously. PPLs, on the other hand have the potential to be a powerful tool in the teaching of programming to VI learners in primary education, combining the physical with the facility to gain an understanding of the overall structure of a program.

3D models (Kane & Bigham, 2014; Papazafiropulos et al., 2016; Stefik et al., 2011) along with auditory and haptic feedback (Calder et al., 2007; Dorsey et al., 2014) have been shown to be useful aids in the teaching process, however they cannot be used to teach programming in isolation and need to be combined with other strategies.

7. Guidelines

Drawing on the literature, a set of guidelines has been produced for educators and developers working with VI learners. It should be noted, however that these guidelines are based on the literature that is currently available and may change as the field develops and more evidence is gathered.

1. Accessible physical programming languages may be a suitable alternative to block-based languages when introducing young VI children to programming.
2. Simple programming concepts can be taught to young VI children using 3D artefacts, for example writing an algorithm to move a bee in a tactile grid.
3. When teaching with text-based programming languages, the choice of language is important. Either choose a language that is specially designed for VI learners, or a general-purpose language with simple syntax and limited use of non-alphanumeric characters, for example Ruby.
4. Ensure you choose a programming environment that is fully accessible and easy to navigate using a screen reader. If an appropriate environment is not available, a plain text editor can be used, although the lack of debugging tools can be challenge.
5. Abstract concepts that are usually taught using visual representations can often be effectively taught to VI learners using 3D artefacts. For example, teaching data structures using different sized cylinders that slot into a tray.
6. VI learners often struggle to gain an overall understanding of the structure of code written in text-based languages, one support strategy is to provide example code in Braille (for braillists).
7. Choosing an appropriate theme for programming activities can make them accessible and engaging for VI learners. For example, tasks that involve programming a physical device, such as a robot can be very engaging. However, it is important to provide positional information for the robot in non-visual forms, this can include the use of auditory and haptic feedback.

References

Albusays, K., & Ludi, S. (2016). Eliciting programming challenges faced by developers with visual impairments. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE '16* (pp. 82–85). Austin, TX, USA. https://doi.org/10.1145/2897586.2897616

Baker, C. M., Milne, L. R., & Ladner, R. E. (2015). StructJumper. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15* (pp. 3043–3052). New York, New York, USA: ACM Press. https://doi.org/10.1145/2702123.2702589

Bigham, J. P., Aller, M. B., Brudvik, J. T., Leung, J. O., Yazzolino, L. a., & Ladner, R. E. (2008). Inspiring blind high school students to pursue computer science with instant messaging chatbots. *ACM SIGCSE Bulletin*, *40*(1), 449. https://doi.org/10.1145/1352322.1352287

Calder, M., Cohen, R. F., Lanzoni, J., Landry, N., Skaff, J., Calder, M., … Skaff, J. (2007). Teaching data structures to students who are blind. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '07* (Vol. 39, p. 87). New York, New York, USA: ACM Press. https://doi.org/10.1145/1268784.1268811

Calder, M., Cohen, R. F., Lanzoni, J., & Xu, Y. (2006). PLUMB: An interface for Users who are Blind to Display, Create, and Modify Graphs. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility - Assets '06* (p. 263). New York, New York, USA: ACM Press. https://doi.org/10.1145/1168987.1169046

Capovilla, D., Krugel, J., & Hubwieser, P. (2013). Teaching Algorithmic Thinking Using Haptic Models for Visually Impaired Students. In *2013 Learning and Teaching in Computing and Engineering* (pp. 167–171). IEEE. https://doi.org/10.1109/LaTiCE.2013.14

Cheong, C. (2010). Coding without sight: Teaching object-oriented java programming to a blind student. In *Eighth Annual Hawaii International Conference on Education* (pp. 1–12). Hawaii International Conference on Education. Retrieved from http://researchbank.rmit.edu.au/view/rmit:13231

Department for Education. (2014). The national curriculum in England - Framework document. Department for Education. Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/381344/Master_final_national_curriculum_28_Nov.pdf

Dorsey, R., Chung, H. P., & Howard, A. (2014). Developing the Capabilities of Blind and Visually Impaired Youth to Build and Program Robots. In *28th Annual International Technology and Persons with Disabilities Conference*. San Diego: California State University, Northridge. Retrieved from http://scholarworks.csun.edu/handle/10211.3/121965

Francioni, J. M., & Smith, A. C. (2002). Computer science accessibility for students with visual disabilities. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education - SIGCSE '02* (Vol. 34, p. 91). New York, New York, USA: ACM Press. https://doi.org/10.1145/563340.563372

Franqueiro, K. G., & Siegfried, R. M. (2006). Designing a scripting language to help the blind program visually. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility - Assets '06* (p. 241). New York, New York, USA: ACM Press. https://doi.org/10.1145/1168987.1169035

Horn, M. S., & Jacob, R. J. K. (2007a). Designing tangible programming languages for classroom use. In *Proceedings of the 1st international conference on Tangible and embedded interaction - TEI '07* (p. 159). New York, New York, USA: ACM Press. https://doi.org/10.1145/1226969.1227003

Horn, M. S., & Jacob, R. J. K. (2007b). Tangible programming in the classroom with tern. In *CHI '07 extended abstracts on Human factors in computing systems - CHI '07* (p. 1965). New York, New York, USA: ACM Press. https://doi.org/10.1145/1240866.1240933

Howard, A. M., Chung Hyuk Park, & Remy, S. (2012). Using Haptic and Auditory Interaction Tools to Engage Students with Visual Impairments in Robot Programming Activities. *IEEE Transactions on Learning Technologies*, *5*(1), 87–95. https://doi.org/10.1109/TLT.2011.28

Jašková, Ľ., & Kaliaková, M. (2014). Programming Microworlds for Visually Impaired Pupils. In G. Futschek & C. Kynigos (Eds.), *Proceedings of the 3rd international constructionism conference*. Vienna. Retrieved from http://constructionism2014.ifs.tuwien.ac.at/papers/2.7_2-8251.pdf

Kane, S. K., & Bigham, J. P. (2014). Tracking @stemxcomet. In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14* (pp. 247–252). New York, New York, USA: ACM Press. https://doi.org/10.1145/2538862.2538975

Konecki, M. (2014). GUIDL as an Aiding Technology in Programming Education of Visually Impaired. *Journal of Computers*, *9*(12), 2816–2821. https://doi.org/10.4304/jcp.9.12.2816-2821

Koushik, V., & Lewis, C. (2016). An Accessible Blocks Language. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS '16* (pp. 317–318). New York, New York, USA: ACM Press. https://doi.org/10.1145/2982142.2982150

Lewis, C. (2014). Work in Progress Report: Nonvisual Visual Programming. In *Proceedings of the 25th Psychology of Programming Annual Conference (PPIG 2014)*. Retrieved from www.ppig.org

Ludi, S. (2013). Robotics Programming Tools for Blind Students. In *28th Annual International Technology and Persons with Disabilities Conference*. San Diego: California State University, Northridge. Retrieved from http://scholarworks.csun.edu/handle/10211.3/121968

Ludi, S. (2015). Position paper: Towards making block-based programming accessible for blind users. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)* (pp. 67–69). IEEE. https://doi.org/10.1109/BLOCKS.2015.7369005

Ludi, S., Ellis, L., & Jordan, S. (2014). An accessible robotics programming environment for visually impaired users. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14* (pp. 237–238). New York, New York, USA: ACM Press. https://doi.org/10.1145/2661334.2661385

Ludi, S., & Reichlmayr, T. (2011). The Use of Robotics to Promote Computing to Pre-College Students with Visual Impairments. *ACM Transactions on Computing Education*, *11*(3), 1–20. https://doi.org/10.1145/2037276.2037284

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Papazafiropulos, N., Fanucci, L., Leporini, B., Pelagatti, S., & Roncella, R. (2016). Haptic Models of Arrays Through 3D Printing for Computer Science Education. In *International Conference on Computers Helping People with Special Needs* (pp. 491–498). Springer, Cham. https://doi.org/10.1007/978-3-319-41264-1_67

Raman, T. V. (1996). Emacspeak---direct speech access. In *Proceedings of the second annual ACM conference on Assistive technologies - Assets '96* (pp. 32–36). New York, New York, USA: ACM Press. https://doi.org/10.1145/228347.228354

Remy, S. L., & L., S. (2013). Extending access to personalized verbal feedback about robots for programming students with visual impairments. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS '13* (pp. 1–2). New York, New York, USA: ACM Press. https://doi.org/10.1145/2513383.2513384

Sánchez, J., & Aguayo, F. (2005). Blind learners programming through audio. In *CHI '05 extended abstracts on Human factors in computing systems - CHI '05* (p. 1769). New York, New York, USA: ACM Press. https://doi.org/10.1145/1056808.1057018

Sánchez, J., & Aguayo, F. (2006). APL: Audio Programming Language for Blind Learners. In K. Miesenberger, J. Klaus, W. L. Zagler, & A. I. Karshmer (Eds.), *Computers Helping People with Special Needs. ICCHP 2006. Lecture Notes in Computer Science* (4061st ed., pp. 1334–1341). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11788713_192

Siegfried, R. M. (2006). Visual programming and the blind: The Challenge and the Opportunity. In *SIGCSE '06 Proceedings of the 37th SIGCSE technical symposium on Computer science education* (Vol. 38, pp. 275–278). Houston, Texas: ACM. https://doi.org/10.1145/1124706.1121427

Siegfried, R. M., Diakoniarakis, D., Franqueiro, K. G., & Jain, A. (2005). Extending a scripting language for visual basic forms. *ACM SIGPLAN Notices*, *40*(11), 37. https://doi.org/10.1145/1107541.1107547

Smith, A. C., Francioni, J. M., & Matzek, S. D. (2000). A Java programming tool for students with visual disabilities. In *Proceedings of the fourth international ACM conference on Assistive technologies - Assets '00* (pp. 142–148). New York, New York, USA: ACM Press.

https://doi.org/10.1145/354324.354356

Stefik, A., Alexander, R., Patterson, R., & Brown, J. (2007). WAD: A Feasibility study using the Wicked Audio Debugger. In *15th IEEE International Conference on Program Comprehension (ICPC '07)* (pp. 69–80). IEEE. https://doi.org/10.1109/ICPC.2007.42

Stefik, A., Hundhausen, C., & Smith, D. (2011). On the design of an educational infrastructure for the blind and visually impaired in computer science. In *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11* (p. 571). New York, New York, USA: ACM Press. https://doi.org/10.1145/1953163.1953323

Stefik, A., & Siebert, S. (2013). An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education*, *13*(4), 1–40. https://doi.org/10.1145/2534973

Stefik, A., Siebert, S., Stefik, M., & Slattery, K. (2011). An empirical comparison of the accuracy rates of novices using the quorum, perl, and randomo programming languages. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools - PLATEAU '11* (p. 3). New York, New York, USA: ACM Press. https://doi.org/10.1145/2089155.2089159

The Royal Society. (2017). *After the reboot: computing education in UK schools*. Retrieved from https://royalsociety.org/~/media/policy/projects/computing-education/computing-education-report.pdf

Thieme, A., Morrison, C., Villar, N., Grayson, M., & Lindley, S. (2017). Enabling Collaboration in Learning Computer Programing Inclusive of Children with Vision Impairments. In *Proceedings of the 2017 Conference on Designing Interactive Systems - DIS '17* (pp. 739–752). New York, New York, USA: ACM Press. https://doi.org/10.1145/3064663.3064689

Appendix A: Literature Cross-Referenced by Theme

	Making Text-Based Languages Accessible	**Making Block-Based Languages Accessible**	**Physical Artefacts**	**Auditory and Haptic Feedback**
(Bigham et al., 2008)	**x**			
(Kane & Bigham, 2014)	**x**		**x**	
(Cheong, 2010)	**x**			
(Smith et al., 2000)	**x**			
(Francioni & Smith, 2002)	**x**			
(Ludi & Reichlmayr, 2011)	**x**		**x**	
(Dorsey et al., 2014)	**x**		**x**	**x**
(Ludi, 2013)	**x**		**x**	
(Ludi et al., 2014)	**x**		**x**	
(Sánchez & Aguayo, 2005)	**x**			
(Sánchez & Aguayo, 2006)	**x**			
(Andreas Stefik, Siebert, et al., 2011)	**x**			
(Andreas Stefik, Hundhausen, et al., 2011)	**x**		**x**	
(Andreas Stefik & Siebert, 2013)	**x**			
(Konecki, 2014)	**x**			
(Baker et al., 2015)	**x**			
(Siegfried, Diakoniarakis, Franqueiro, & Jain, 2005)	**x**			
(Franqueiro & Siegfried, 2006)	**x**			
(Siegfried, 2006)	**x**			
(A. Stefik et al., 2007)	**x**			
(Lewis, 2014)		**x**		
(Ludi, 2015)		**x**		
(Koushik & Lewis, 2016)		**x**		
(Thieme et al., 2017)			**x**	
(Papazafiropulos et al., 2016)			**x**	
(Capovilla et al., 2013)			**x**	
(Calder et al., 2007)				**x**
(Calder et al., 2006)				**x**
(Howard, Chung Hyuk Park, & Remy, 2012)	**x**		**x**	**x**
(Jašková & Kaliaková, 2014)		**x**	**x**	**x**
(Remy & L., 2013)			**x**	

Exploring the Use of Self-Regulation Strategies in Programming with Regard to Learning Styles

Ünal Çakiroğlu [1]
Betül Er[2]
Nursel Uğur[3]
Esra Aydoğdu[3]

[1]Karadeniz Technical University
[2]Agri University
[3]Ministry of Education

DOI: 10.21585/ijcses.v2i2.29

Abstract

This study attempts to understand the relationship between learning styles and self-regulated learning of pre-service computer teachers in a programming course. Students' strategies for self-regulation with regard to their learning styles were assessed on the basis of qualitative data in terms of programming course. The Turkish version of Felder-Soloman learning style inventory was used to identify the students' learning styles and interviews were conducted to evaluate students' SRL strategies in programming. The results suggest that the characteristics of learning styles are somewhat related to self-regulation strategies. Time management was identified as a leading self-regulation strategy among learning styles, while shortcomings regarding target setting and self-efficacy strategies were prominent with almost all learning styles. Characteristics of other self-regulation strategies do not directly match with expected behaviors of learning styles in the context of learning programming. It is hoped that the study may shed light for instructors and instructional designers to design more appropriate settings for teaching programming taking learning styles in to consideration.

Keywords: self-regulated learning, learning styles, programming

1. Introduction

Computer programming is considered as a challenging course given the extensive set of knowledge and skills through the years (Bennedsen and Caspersen, 2008). Researchers often addressed that students struggle in the transition from introductory level programming to more advanced level. This is because in programming learning processes, students need to use various cognitive and metacognitive strategies to control and regulate their own learning (Brennan and Resnick, 2012; Hwang, Liang, and Wang, 2016). Numerous studies indicated that learning programming cannot be confined in the classroom only, and emphasize the need for applicable work for outside the classroom (Azevedo and Hadwin, 2005; Kozlowski and Bell, 2006; Wiedenbeck, LaBelle and Kain, 2004). Accordingly, investigating metacognitive processes underlying the learning programming has gained more attention. Thus, prior work in teaching programming pedagogy has focused on some problem solving strategies and techniques for overcoming difficulties in teaching programming (Lau and Yuen, 2011; Nam, Kim and Lee, 2010; Saeli et al., 2011).

In this circumstance, some researchers address greater involvement in the learning process and students' responsibility in their learning (Akpınar and Altun, 2014; Lye and Koh, 2014). In order to manage their learning; Self-regulated Learning (SRL) enables students to be active in the learning process while developing programming skills (Zimmerman, 2008). In this circumstance, specific SRL strategies are required to perform the programming tasks (Armstrong, 1989). It is evident that students who use these strategies can perform high in the process of learning programming (Alharbi et al., 2014; Falkner et al., 2014). Since programming requires

self-regulation, many efforts are ongoing about developing self-regulation for programming. In this context, perspectives about how SRL arise in learning programming are discussed in the following section.

1.1. Self-Regulation in Programming

SRL allows students to be active and to direct their learning (Fernández et al., 2012; Zimmerman, 2002). Researchers have reached a consensus that students' SRL strategies have been positively related to their achievements (Artino, 2008; Artino, 2009; Lee, Shen, and Tsai, 2010; Liaw and Huang, 2013; Paechter, Maier and Macher, 2010; Pintrich, 2000; Puzziferro, 2008; Wang, Shannon and Ross, 2013). In general, the activities in the learning processes are considered as mediators between students, contexts, and achievement within SRL strategies (Pintrich, 2004). Safari and Hejazi (2017) argued that self-regulated learners can get advantage of their own learning because they know how to apply the acceptable actions in order to reach the goals. In the educational context; SRL strategies are seen in the dimensions of self-evaluation, organization, and transformation; goal setting and planning; seeking information, keeping records, and monitoring; environment structuring; self-consequences; rehearsing and memorizing; seeking social assistance; and reviewing records are used (Pintrich and DeGroot, 1990; Schunk and Zimmerman, 1998; Zimmerman, 2002; Zimmerman and Martinez-Pons, 1990).

In programming learning domain self-regulation arguably plays a key role in facilitating the development of major skills of problem solving such as logical thinking and reasoning, and helps students to manage their learning process during programming (Ramalingam, LaBelle and Wiedenbeck (2004). Some studies suggest that, self-regulated learners can find a number of ways to achieve the goals in programming learning process (Bergin, Reilly and Traynor 2005; Kumar et al. (2005). In this sense, some recent studies suggest planning, self-evaluation and self-monitoring (Falkner, Vivian, and Falkner, 2014; Falkner, Szabo, Vivian, and Falkner, 2015; Li, Ko, and Zhu, 2015) are prominent strategies to achieve learning objectives in programming. Also, self-efficacy is considered as one of the main strategies of SRL which keeps students on track in learning programming (Kuo, Wu and Lin, 2013; Ramalingam et al., 2004; Wiedenbeck, 2005). Self-satisfaction is another factor which is emphasized by Kuo, Wu and Lin (2013) in their model. In addition, a recent study suggests that designing instruction through self-regulation skills for programming courses enhances problem solving skills (Loksa et al., 2016).

On the other hand; Hui and Umar (2010) highlighted some individual characteristics, such as learning styles and Wiedenbeck (2005) addressed previous programming experience and knowledge of organization were also important for learning programming. The fact that learning styles as processing information can certainly affect students' progress and their programming performances. In this context, the recent studies frequently reference to learning styles to understand the progression in learning programming.

1.2. Learning Styles in Programming

Safari and Hejazi (2017) point out that one of the learning obstacles in classroom is the lack of coordination between the instructional methods and the learning styles. Learning styles which are related to the way of students' information processing skills may influence to the student's performance in introductory programming' (Norwawi, Abdusalam, Hibadullah, and Shuaibu; 2009). When students are aware of their own different styles, they can learn better. Research studies indicate that matching learning styles with teaching methods provide high academic achievements. For instance, Alharbi et al. (2011) reported that some of the students in computer science programs are not aware of their SRL, and that they do not know how to apply SRL strategies in the learning process. Some other studies indicated that students with different learning styles prefer to use different SRL strategies (Shannon, 2008) and stating a relationship between SRL and learning styles may promote learning on the part of students (Safari and Hejazi, 2017). It can be considered that in the process of learning programming, students with different learning styles may follow or develop different self-regulation strategies. Thus, instructors should create authentic learning environments by being familiar with individual students' learning styles.

1.3. Considering SRL with regard to Learning Styles

It is known that student- centred educational paradigms place a high level of responsibility on learners to control and regulate their personal learning processes. It is also crucial to take individual differences into consideration in instructional processes (Das, 2015). Emphasizing the responsibility students' own learning, Paris and

Winograd (2001) suggest promoting self-regulatory learning strategies. Being aware of the students' learning styles, teachers could help them to know their learning habits, and help them to apply better learning strategies within this responsibility. Since in problem solving in problem solving process acquired by self-regulated strategies (Zhang et al,2006), learning style as the characteristic cognitive, affective and psychological behaviors may serve as how learners perceive, interact with and respond to the learning environment (Keefe, 1988). Thus, within the student-centred paradigm, understanding students' preferences and the self-regulated learning strategies together may facilitate their learning process. In a study focusing on the relationships between SRL and learning style, Man-Chih, (2006) documented that since self-regulation provides learners with a role in decision-making; it is in an accord with converging learners' styles. Lavasani et al (2011) also found self-regulated learners using metacognitive strategies to get advantage of learning process which is in line with the feature of the diverging learners. Gülbahar (2005) referring to the SRL, argued that any student can adapt learning processes, activities and techniques, if he/she is able to understand his/her own learning styles and also be aware of his experiences.

1.4. Aim of the Study

While prior work has investigated many aspects of programming in terms of the role of self-regulation, more detailed investigation of the relationship between self-regulated learning and learning styles in the field of programming is needed. In order to facilitate learning, instructors should provide an easy way for students to discover their own characteristics. Thus, exploring the self-regulatory mechanisms regarding the learning styles would reveal the nature of psychological processes essential to the initiation, maintenance, and may be termination of learning in programming. So, this paper is hoped to contribute to understanding the relationships between learning styles and self-regulated learning strategies in the field of programming in higher education.

In line with the overall purpose of the study, the following research question was directed: How students' self-regulation strategies differentiate in terms of their learning styles in the context of programming learning process?

2. Method

A Turkish version of the Felder-Soloman's learning style inventory (LSI-T) coupled with a semi structured interview was applied to answer the research questions. The results from the interviews were then categorized and interpreted regarding the students' learning styles.

2.1. Participants

The study was carried out in a programming language course at the Computer and Instructional Technologies Department of a major university in Turkey. The participants were 57 pre-service computer teachers (29 male, 28 female) between the ages of 18 and 24. The participants have basic computer literacy skills, and limited prior programming knowledge. It was the first time they were receiving an introductory programming course. After, determining the learning styles of all students, 8 students (4 male 4 female) from all learning styles were interviewed.

2.2. Instrumentation

2.2.1. LSI-T

To identify students' learning styles, the Turkish language version of Felder and Soloman's Index of Learning Styles inventory was administered. Felder-Soloman index is one of the most widely used inventories in teaching programming, reflecting the skills required for learning programming. Felder and Soloman (1998) in their learning style model categorized learners according to four main characteristics, and classified the learning styles as: active-reflective; sensing-intuitive; visual-verbal; and sequential-global. While active learners prefer learning by doing or actively participating in work and prefer social interaction, reflective learners think about the task first. It means they prefer thinking quietly about information rather than be interactively engaged in learning activities. Intuitive learners would be more comfortable managing their own learning, so they prefer finding learning possibilities, discovery, innovation, and abstractions. In contrast, sensing learners deal with facts and concepts, example-based, concrete learning (Dille and Mezack 1991). In addition, verbal learners get more

out of words than from visual representations and the global learners learn in large jumps by seeking out the "big picture" rather than learning in the traditional, sequentially organized college course.

The Index of Learning Styles (ILS) (Felder and Soloman, 1991) is a 44-question survey based on a learning style model. The validity and reliability of the index have been verified by a number of studies (Felder and Spurlin, 2005; Litzinger et al., 2007; Zywno, 2003). One reason selecting this inventory is the potential of the inventory for determining whether the learner has a strong, moderate, or low preference on the identifiers of the learner styles. The index was translated into Turkish and validity and reliability analyses were provided (Büyüköztürk, Akgün, Özkahveci and Demirel, 2004). The LSI consists of 44 two-part ('a' and 'b') items. Each item comes with two options, where 'a' represents active, sensing, visual, sequential learning styles while 'b' suggests reflective, intuitive, verbal and global ones.

The inventory was used in previous programming teaching studies. For instance, Chen and Lin (2011) used the inventory to identify learning styles at the beginning of programming instruction. In a similar vein, Norwawi Abdusalam and Hibadullah (2009) applied the Felder-Soloman learning styles inventory with master's students prior to the beginning of the course.

2.2.2. Interviews

Semi-structured interviews were conducted to gather the perceptions about students' SRL strategies. The interview questions were developed on the basis of the strategies referring to the SRL definitions (Pintrich, Smith, Garcia and McKeachie, 1991). The strategies were in relation with task value, external target orientation, target setting, self-efficacy, self-reflection, repetition, peer learning, time management, and effort regulation. When formulating the semi-structured interview questions, various SRL scales focusing on these strategies were also reviewed, within the framework of the programming languages course. The selected interview questions are presented in Appendix 1.

2.3. Data Analysis

To find mean scores for each learning styles, instances where option 'a' was chosen were coded 1, and the instances with option 'b' were coded 2. Referring to the original inventory mean scores in the 11 to 16 range represent active, sensing, visual and sequential learners. On the other hand, mean scores in the 17 to 22 range represent reflective, intuitive, verbal, and global learners (Arslan and Aksu, 2006).

The data obtained through the interviews, in turn, were analyzed through content analysis. The interviews were transcribed into text, followed by thematic analysis based on expressions in common statements to define the main themes. First, initial codes were identified by two coders. After examining the responses, the coders produced tentative thematic units. Thereafter relationships, similarities and differences between the codes assigned by both coders were reviewed and categorized, culminating in the construction of the themes in a manner to ensure perfect concurrence among the coders, regarding the final themes. Moreover, direct quotations in association with the learning styles were presented with special attention being paid to maintain the meaning.

3. Results

The results are presented regarding the relationship between the learning styles and the use of self-regulation strategies. In this context, a two-dimensional tabular presentation is used in order to express the codes related to the strategies reflecting learning styles. Participants are assigned as S_n according to their learning styles. The interviewed students' perceptions about the SRL strategies in the context of the programming course were presented in line with their different learning styles. The perceptions about the strategies used in "task value" category are shown in Table1.

Table 1. Strategies in task value category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Task Value	Making practical use of the profession		√	√	√	√		√	
	Providing associations with various fields	√	√	√	√	√	√	√	
	Considering it as a means to enhance intelligence	√		√					
	Not attaching value					√			√

Table 1 show that the majority of the students believe that; they will use programming experience in their professional life, and they consider programming as the one of the basic competencies of being a computer teacher. All students noted that programming logic is related with a number of distinct fields. A participant with intuitive style expressed this point by saying "*It can be used in a number of fields. We use the logic unwittingly, even in our daily lives*". Only the students with active and sensing learning styles deemed the programming course as a means to enhance intelligence, whereas those of visual and global styles did not address any task-value point with respect to the programming course.

Students' views regarding the "external target orientation" are presented in Table 2.

Table 2. Strategies in external target orientation category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
External Target Orientation	Passing the course	√	√		√		√	√	√
	Engaging in high-level programming		√					√	
	Learning the basics	√		√		√			
	Developing programs with no support		√						

A glance at the students' aims and intentions associated with the programming course indicate that only sensing and visual students intended to pass the course. Indeed, the ones with intuitive, verbal, and global styles expressed that passing the course is their main aim with the course. The ones with reflective and sequential style students added the ability to engage in high-level programming. In addition the active, sensing, and visual style students noted the expectation to learning the basics of programming as well. The student with the reflective style, in turn, mentioned the importance of writing programs without getting help.

Strategies expressed with the "target setting learning" are summarized in Table 3.

Table 3. Strategies in Target Setting Category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Target Setting	Completing the program			√	√	√	√		√
	Getting ahead of the peers	√					√		
	Developing different solutions						√		
	Comprehending programming logic	√							
	Setting no target							√	

A substantial number of the students (sensing, intuitive, visual, verbal, global) intend to complete the program they are writing, without any bugs. Furthermore, the students with verbal style include getting ahead of the peers and developing unique solutions. Those with the active style intend to grasp at least the logic of the program. Sequential style student stated that she have no targets. Such targets set by students before venturing with the program usually prevent dropping out of the endeavor prematurely.

Self-efficacy is considered one of the most important category concerning self-regulating learning. In this context, the students' perspectives about the strategies related to self-efficacy are expressed in Table 4.

Table 4. Strategies in self-efficacy category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Self-Efficacy	Providing a holistic view for the problem								√
	Employing distinct solutions		√				√		
	Employing common solutions	√							
	Writing brief and easy-to-decipher programs				√	√	√		
	Adding a visual perspective				√				

Some students with global style had a preference for holistic approaches. Moreover, those with reflective and verbal styles emphasized confidence in their ability to find distinctive solutions to the problem, while those with the active style noted the advantages of applying common solutions. Those with intuitive, visual, and verbal styles expressed their ability to write brief and comprehensible programs, while those with the intuitive style exclusively referred to the ability to add a visual element. In this sense, S4 expressed that "*If the program is about a ball, the color of that ball is important for me.*" The ones with the visual style, on the other hand, voiced

their confidence in perseverance in the face of problems. The student with the sensing style surprisingly did not note any strategy regarding self-efficacy. Students' views with respect to self-reflection are presented in Table 5.

Table 5. Strategies in Self-Reflection Category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Self-Reflection	Reflecting bugs incurred in the program	√	√	√	√	√	√	√	√
	Forgetting programming approaches	√		√					√
	Difficulties in directing attention		√						
	Not taking time for programming					√	√		
	Getting bored			√			√		
	Checking incorrect codes		√	√			√		√
	Repeating	√		√					
	Getting help	√		√	√	√	√	√	√
	Trying different ways			√		√			
	Developing ambition	√							

All students noted that they experienced at least one bug when writing programs and those they checked the code to correct them. Those with a style other than reflective, on the other hand, expressed that they got help. The one with the sensing style expressed that "*I often faced with errors. To overcome them, I either have to check the bugs or I get help from my friends.*" Other students with active, sensing, and global styles addressed that they sometimes have difficulty in how to find appropriate programming approaches. To overcome this problem, they noted the need to repeat certain structures. Visual and verbal style students on the other hand, referred to the inability to take time required for programming, as the leading problem they faced, while the ones with sensing or verbal styles expressed that they got bored when writing programs. In addition, the active style student expressed that she would get rather avid as she noticed the shortcomings, while the one with the sensing style noted her competence about trying different means to solutions. The views voiced with respect to repetition and peer learning strategies are expressed in Table 6 and Table 7 respectively.

Table 6. Strategies in repetition category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Repetition	Taking notes about important pieces of code		√		√				√
	Developing solutions on paper	√	√	√			√		√
	Reviewing sample programs	√				√		√	√
	Trial and error on a computer				√	√		√	
	Developing and solving similar questions	√							
	Memorizing			√		√			

When preparing for the exam, half of the students who have distinctive learning styles (active, visual, sequential, global) reviewed sample programs and took notes about the scripts. In this context, the student with visual style specified that "*First of all, I check the sample applications and learn about the common forms. Then I try to write the software to understand the overall logic.*" Students with intuitive, visual, and sequential styles studied on exercises to get ready for the exams, while the rest prefer writing codes on paper. The student with active style said that she developed similar questions and tried to solve them. Those with sensing and visual styles memorized the pieces of scripts to prepare for the exams.

Table 7. Strategies in peer-learning category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Peer Learning	Providing solutions for errors	√	√	√	√		√		√
	Doing homework		√						
	Studying for exams	√						√	√
	Required no help		√			√			

The students with reflective and visual learning styles addressed that they tried to refrain from getting help; yet they often get help from their peers in their efforts to solve the problems they often encounter. In addition, the students cooperate with their peers for doing homework as with the reflective style student noted, and during exam weeks as active sequential, and global style students mentioned.

Time-management of self-regulated strategies is generally associated with the classroom activities as well. In this context, the perspectives regarding the time-management are presented in Table 8.

Table 8. Strategies in time management category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Time Management	Not spending enough time	√	√	√	√		√	√	√
	No planning					√	√	√	√
	Completing assignments after the class		√		√				
	Performing assignments late	√		√			√		
	Preparing for the exams in last day		√			√	√		√

The students except the one with visual style noted insufficiency of the time devoted to the programming course. The ones with visual, verbal, sequential, and global styles, in particular, pointed out the lack of any planning for time-management for this course. The ones with reflective and intuitive styles completed the assignments right after class, while the ones with active, sensing, and verbal styles had a preference for waiting for the last period. Students with reflective, visual, verbal, and global styles addressed that they studied for the programming exams only on the last day before the exam. Participant having global learning style objective was only to pass the course, and expressed by commenting "*I study for the exams on the very last day. It allows me to get a grade of 45.*" Views about "effort regulation" are shown in Table 9.

Table 9. Strategies in effort regulation category

		Students' Learning Styles							
	Strategies	S1.Active	S2.Reflective	S3.Sensing	S4.Intuitive	S5.Visual	S6.Verbal	S7.Sequential	S8.Global
Effort Regulation	Having a break	√	√	√	√	√	√		√
	Seeking help			√	√		√		
	Giving up completely							√	
	Getting motivated only with the easier parts						√		

Most of the interviewees stated that they usually took a break with programming when they face any challenges to resume the efforts later on, while the one with the sequential style expressed that she completely gave up in such cases. She pointed out this by saying "*I demotivated and give up if I fail to solve a problem.*" Moreover, the ones with the sensing, intuitive, and verbal learning styles noted getting help to overcome issues. The verbal style student expressed that she would be motivated by handling the easier parts of the program first, which would motivate her for the rest of the problem. The leading strategies employed by students who have different self-regulating learning strategies for programming course with regard to Felder and Soloman (1994) inventory are summarized in Figure 1. The figure was developed on the basis of positive perspectives with respect to a substantial portion of indicators concerning a given strategy. Overall, Figure 1 summarizes the perceptions about the self-regulation strategies employed by students with different learning styles in the context of learning programming. Also examples from active and sequential style students' artifacts are provided in Appendix2.

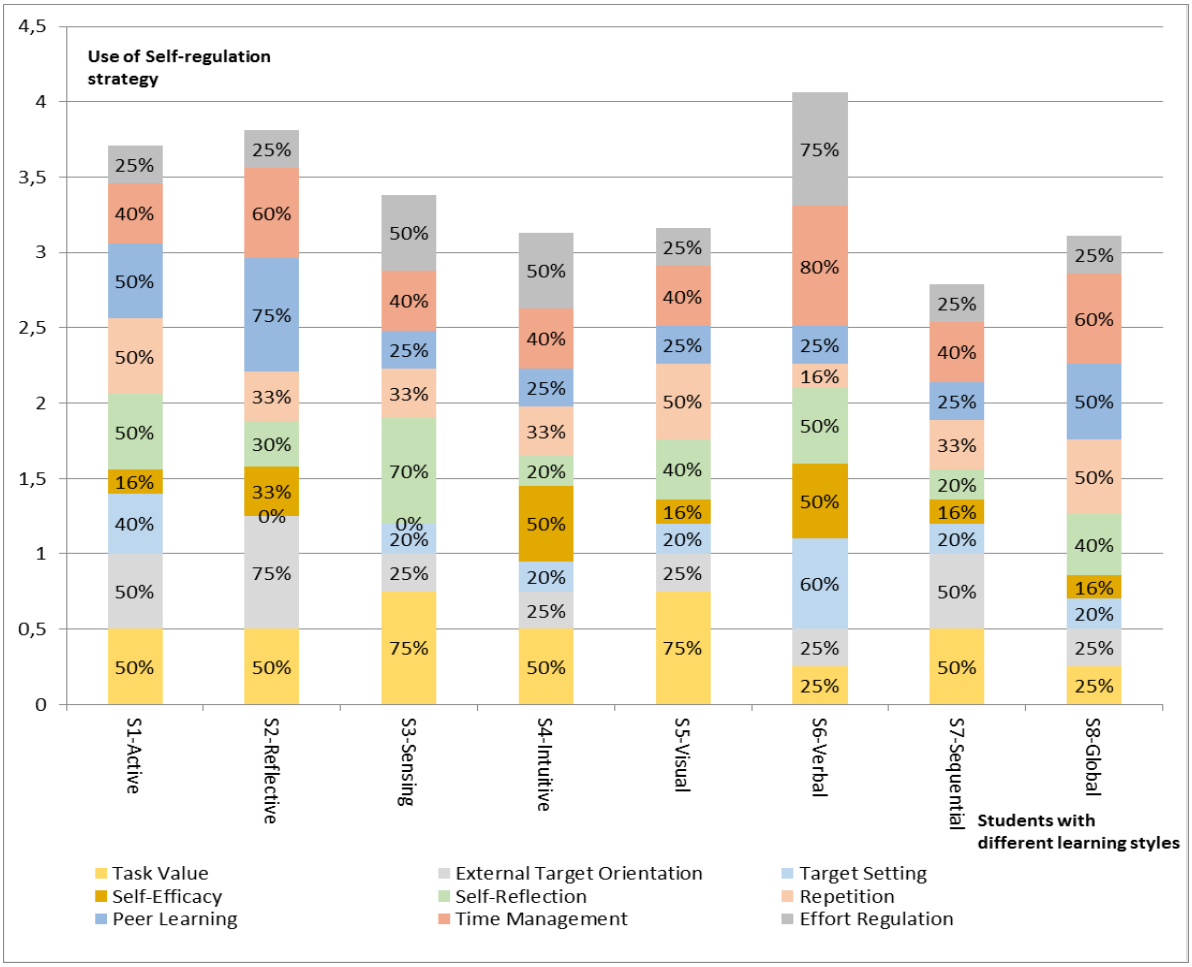

Figure 1. Use of self-regulating learning strategies with reference to their learning styles

Students' perceptions with regard to self-regulated learning strategies indicate that various learning styles are prominent in using some strategies. Yet, the ones with visual and sequential learning styles expressed clues about a rather limited set of indicators compared to the expressions of other students, while the ones with active and verbal styles voiced concrete statements concerning numerous indicators. Figure 1 reveals that students with visual, verbal, and global styles did not express any statements about the task value strategy, while the ones other than those with the active, reflective, and sequential strategies expressed that they somewhat provide external target orientation strategies. Furthermore, the target setting and self-efficacy strategies stand out as the ones where the concrete statements on part of the students were rarest. Students with active, sensing, and verbal learning styles expressed positive views about the self-reflection strategy, while students with other learning styles voiced positive views was only about a very limited set of items. On the other hand, participants with active, visual and global styles perceptions regarding the repetition strategy were explicitly positive. Students

with active, reflective, and global strategies expressed clues concerning numerous indicators associated with the peer-learning strategy.

4. Discussion

Considering prior work, this study is based on the idea that students with various individual characteristics can exhibit different self-regulation strategies during learning programming. Felder-Soloman learning styles inventory is generally discussed with reference to four dimensions considering the similarities among the styles: sensing/ intuitive, visual/ verbal, active/ reflective, sequential/ global. The result of this study indicated that peer-learning and external target orientation were prominent strategies in reflective learning style, while the sensing style students emphasized task value and self-reflection strategies. In the same vein, Das (2015) found a significant relationship between self regulated learning and cognitive styles. Students having different learning styles perceived in various extend of self-regulated strategies in this study. Relationships between the characteristics of learning styles and t their perceptions about the self-regulated strategies are discussed in the following section.

4.1. Sequential / Global

Students with a sequential learning style learn the knowledge offered, as a sequence of interrelated smaller parts. Global learners generally need to associate new knowledge with their preliminary knowledge and experience, before getting acquainted with the details of the topic. Sequential learners, on the other hand, can utilize specific details without embracing the topic as a whole. But they can have problems in grasping the connections the topic has with other fields and disciplines (Felder and Silverman, 1988). In the present study, student with sequential learning style expressed an emphasis on task value and external target orientation strategies. Students who employ a sequential learning style can associate the knowledge with their pre-existing knowledge. Hence, it is only natural that the task value and external target orientation strategies are emphasized by students who have sequential learning style, for whom motivation is considered a substantial factor in terms of self-regulating learning strategies employed. Student having a global learning style, in her turn, had emphasized the repetition, peer-learning and time management strategies. Also, students who have a global learning style can achieve lasting learning by associating the new knowledge with their previous experiences. One can find an emphasis on the repetition strategy curious for a student with global style. The repetition strategy which is based on memorizing and global learning style is all about associating meaning through experience.

4.2. Active/Reflective

Active learning process entails interactions with the external world, such as discussing, expressing, or testing the acquired knowledge. Active learners prefer to be engaged in physical activity, while the reflective ones choose contemplation about knowledge offered (Felder and Silverman, 1988). In the study, the student who had active learning style was positively perceived task value, external target orientation, self-reflection, repetition, and peer learning self-regulating learning strategies. Since active learning style is often about considerations with the external world, his emphasis on the external target orientation and peer learning strategies are not surprising, through the characteristics of this style. In addition, student who had reflective learning style highlighted the task value, external target orientation, and peer learning self-regulating learning strategies. Taking the emphasized strategies into consideration, one can refer to a positive relation between the learning style and time management, particularly in the light of the results observed with student with reflective learning style.

4.3. Visual / Verbal

According to Felder and Silverman (1988), the visual learners prefer to use visual elements such as images, diagrams, schemes and presentations for the presentation of knowledge, compared to verbal statements or written texts, while the verbal learners prefer verbal statements and written texts. In this study the student who had a visual learning style had also found the task value and repetition self-regulating learning strategies important. The students with a visual learning style do not need the repetition strategy to the extent the students with a verbal style do. In the same vein, the visual learner's emphasis on the repetition self-regulating learning strategy is considered to be surprising. Verbal learning style student emphasized target setting, self-efficacy, self-reflection, time management, and effort regulation strategies. Thus, it can be concluded that that time management and regulation strategies could be expected, while the repetition strategy was once again a surprise.

4.4. Sensing / Intuitive

Felder and Silverman (1988) in their learning style model defines sensing learning style having a preference for the knowledge directed at their senses and the intuitive learners are better with knowledge arising internally, from their own ideas. Student who had a sensing learning style in this study had addressed task value, self-reflection, and effort regulation self-regulating learning strategies. Students with sensing learning style are inclined to receive the knowledge directed towards their senses. In this context, emphasis on the effort regulation and task value strategies on part of the student with the sensing learning style may be a function of the perception with reference to her learning style. The student who had an intuitive learning style had emphasized the task value, self-efficacy, and effort regulation self-regulating learning strategies. The intuitive learners welcome the knowledge they can imagine internally in their minds, while the students with the intuitive learning style presented with a surprise in the form of the emphasis on effort regulation self-regulating learning strategy.

Overall, students with the active, reflective, and verbal learning styles presented different self-regulation strategies. These students evidently focus more on the self-regulation strategies, compared to the adherents of other learning styles. The perceived self-regulation strategies among students who have active, intuitive, and sequential learning styles does not lead to substantial differences. In terms of the programming learning process, planning is pervasive throughout problem solving, guiding the direction that programmers take. Few participants were expected to exhibit planning, given their inexperience. The lack of emphasis on planning with respect to time management in particular, on part of students of some learning styles (visual/ verbal, sequential/ global) is noteworthy. In the present study, students with a wide range of learning styles were found to exhibit skills such as making associations with various fields, passing the course which can be listed among the motivational factors. The similar results reached by Das (2015) revealed a significant relationship between self regulated learning and cognitive styles of secondary school students. There is also a consistency between the findings of this study and prior studies about the relationship between learning styles and self-regulation strategies. For instance, the study by Kumar et al. (2005) indicates that programmers who received self-regulated-based treatment outperformed those who did not. Safari and Hejazi (2017) examined the relationship between the Kolb learning styles –converging, diverging, assimilating, and accommodating– with the participants' self-regulation skills. The results showed a positive relationship between each learning styles and self-regulation skills.

The study has also some limitations. The study was carried out with a limited size of sample selected from those who received the Programming Languages course. With a sample size of 8 it was difficult to generalize the power needed to precisely identify relations. Yet, the data gathered allow us to reach to certain conclusions regarding the role of students' work and their individual characteristics in a qualitative manner. The data from interviews provided the discussions about the nature of the strategies.

5. Conclusion and Recommendations

The analysis sought to reveal the association between the learning style and the self-regulated learning in terms of learning programming, with reference to the views expressed by students. The results suggest that learning styles are somewhat related to self-regulation strategies. In particular, perceptions of the students with active/reflective and sequential/global learning styles with reference to their SRL strategies considerably reflect the characteristics of those learning styles.

On the other hand, students with visual/verbal learning styles did not strongly reflect those characteristics. Some of the students in visual/verbal learning styles in turn, surprisingly expressed perceptions which are deemed to be in contrast to those expected of their learning styles. Time management was identified as a leading strategy among learning styles, while shortcomings regarding target setting and self-efficacy were prominent with almost all learning styles. Thus, one can argue that learning styles and perceptions about SRL strategies are somewhat related to each other. Nonetheless, some external factors may still affect this relationship. In this sense, in the process of teaching programming, it is observed that directing students' SRL strategies is not an easy task. In other words, the reflections of students' learning styles may not always match with their strategy use in the learning process. In such cases, programming instructors may be compelled to find new ways for customizing the learning process.

The present study investigated differentiation of learning styles with reference to qualitative interview data. This allowed the association of the characteristics of learning styles, and the perceptions about the basic indicators of self-regulation. Despite these and many other limitations affecting to the generalizability of our results, the

results may still be considered as a first step in understanding the relationship between learning styles and self-regulation in programming. Since SRL strategies offer the potential of facilitating learning programming; the results of this study suggest taking note of individual characteristics to inform the application of the strategies is noteworthy. We hope that the results provide some insights into the self-regulated learning in a higher education programming instruction.

References

Akpınar, Y., Altun, Y. (2014). The need for programming education at the schools of an information society. *Elementary Education Online*, *13(1)*, 1-4.

Alharbi, A., Henskens, F., and Hannaford, M. (2014). Personalised learning object system based on self-regulated learning theories. *International Journal of Engineering Pedagogy*, *4(3),* 24-35. http://dx.doi.org/10.3991/ijep.v4i3.3348

Alharbi, A., Paul, D., Henskens, F., and Hannaford, M. (2011). An investigation into the learning styles and self-regulated learning strategies for computer science students. *In Proceedings Ascilite. Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS),* 2008.

Armstrong, A. (1989). The development of self-regulation skills through the modeling and structuring of computer programming. *Educational Technology Research and Development*, *37(2)*, 69-76.

Arslan, B., Aksu, M. (2006). A descriptive study on the learning style profiles of engineering students of the Middle East Technical University (METU). *Education and Science, (31)*, 141.

Artino, A.R. (2008). Motivational beliefs and perceptions of instructional quality: Predicting satisfaction with online training. *Journal of Computer Assisted Learning, (24)*, 260-270.

Artino, A.R. (2009). Think, feel, act: motivational and emotional influences on military students' online academic success. *Journal of Computing in Higher Education,(21),* 146-166.

Azevedo, R.., Hadvin, A. (2005). Scaffolding self-regulated learning and metacognition-implications for the design of computer-based scaffolds. *Instructional Science, (33),* 367-379.

Bennedsen, J. and Carpersen, M. E. (2008). Exposing the programming process. Bennedsen, J.,Carpersen, M. E. ve Kolling, M. (Eds.). *Reflection on the theory of programming: Methods and implementation*, 6-16. New York: Springer Berlin Heidelberg.

Bergin, S., Ronan R., Desmond, T. (2005). Examining the role of self-regulated learning on introductory programming performance. *In Proceedings of the First International Workshop on Computing Education Research, (pp.81-86)*. Seattle, WA, USA.

Brennan, K., Resnick, M. (2012). Using artifact-based interviews to study the development of computational thinking in interactive media design. *Paper presented at annual American Educational Research Association meeting*, Vancouver, BC, Canada.

Büyüköztürk, Ş., Akgün, Ö. E., Özkahveci, Ö., and Demirel, F. (2004) .Validity and reliability study of Turkish version of the scale of motivation and learning strategies. *Theory and Practice in Educational Sciences,4(2),* 207-239.

Chen, C. L., & Lin, J. M. C. (2011). Learning styles and student performance in java programming courses. *In Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS) (p. 1)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Das, A. (2015). Self Regulated Learning and Cognitive Styles of School Students - A Study, *International Journal of Science and Research*, *5(12),* 1691-1694

Dille, B., Mezack, M. (1991). Identifying Predictors of High Risk among Community College Telecourse Students, *The American Journal of Distance Education, 5(1),* 11-19.

Falkner, K., Szabo, C., Vivian, R. and Falkner, N. (2015). Evolution of software development Strategies. *In Proceedings of the 37th International Conference on Software Engineering.* Florence, Italy.

Falkner, K., Vivian, R. and Falkner, N. (2014). Identifying Computer Science Self-Regulated Learning Strategies. *In Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, 2, pp.* 291-296.

Felder, R.M. & Silverman, L.K. (1988). Learning and teaching styles in engineering education. *Engineering Education, 78(7),* 674-681.

Felder, R.M. & Soloman, B. A. (1994). Index of Learning Styles. North Carolina State University, Retrieved January 06, 2017, from http://www.ncsu.edu/felder-public/ILSdir.html.

Felder, R.M., and Spurlin (2005). Applications, Reliability and Validity of the Index of Learning Styles. *International Journal of Engineering Education, 21(1),* 103-112.

Gülbahar, Y. (2005). Learning Styles and Technology, *Education and Science, 30(138)*, 10-17.

Hui, H., Umar, I. (2011). Pair programming and lss in computing education: its impact on students' performances. *US-China Education Review, 8*(2), 613-626.

Hwang, G., Liang, Z. and Wang, H. (2016). An Online Peer Assessment-Based Programming Approach to Improving Students' Programming Knowledge and Skills. *In Educational Innovation through Technology (EITT), 2016 International Conference on* (pp. 81-85). Tainan, Tayvan: IEEE.

Kfee, B.J. (1988). The logic of message design: Individual differences in reasoning about communication. *Communication Monographs, 55*, 80-103.

Kozlowski, S., Bell, B. (2006). Disentangling achievement orientation and goal setting: effects on self-regulatory processes. *Journal of Applied Psychology*, 900-916.

Kumar, V., Winne, P. H., Hadwin, A. F., Nesbit, J. C., Jamieson-Noel, D., Calvert, T., et al. (2005). Effects of self-regulated learning in programming. *In Fifth IEEE International conference on advanced learning technologies (ICALT 2005) (pp. 383-387).* IEEE.

Kuo, F., Wu, W. and Lin, C. (2013). An investıgatıon of self-regulatory mechanisms in learning to program visual basic. *Journal Educational Computing Research, 49*(2), 225-247.

Lau, W., Yuen, A. (2011). Modeling programming performance: beyond the influence of learner characteristics. *Computers & Education, 571*(1), 1202-1213.

Lavasani, M. G., Mirhosseini, F.S., Hejazi, E., Davoodi, M. (2011). The Effect of Self-regulation Learning Strategies Training on the Academic Motivation and Self-efficacy. *Procedia Social and Behavioral Sciences, 29*, 627 – 632.

Lee, T.H., Shen, P.D., & Tsai, C.W. (2010). Enhance students' computing skills via webmediated self-regulated learning with feedback in blended environment. *International Journal of Technology and Human Interaction, 6(1),* 15-32.

Li, P., Ko, A. and Zhu, J. (2015). What Makes a Great software Engineer? *In Proceedings of the 37th International Conference on Software Engineering*, Florence, Italy.

Liaw, S.S. & Huang, H.M. (2013). Perceived satisfaction, perceived usefulness and interactive learning environments as predictors to self-regulation in e-learning environments. *Computers & Education, 60(1),* 14-24.

Litzinger, T.A., Lee, S.H.; Wise, J.C., and Felder, R.M. (2007). A Psychometric Study of the Index of Learning Styles. *Journal of Engineering Education, 96(4),* 309-319.

Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, problem solving, and self-awareness: effects of explicit guidance. I*n Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (pp. 1449-1461). ACM.

Lye, S. Y., Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior, 41,* 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Man-Chih, A. (2006). *The effect of the use of self-regulated learning strategies on college students' performance and satisfaction in physical education* (Unpublished Doctoral Dissertation). Australian Catholic University, Australia. Retrieved March 06, 2018, from https://doi.org/10.4226/66/5a94b7585e4cb

Nam, D., Kim, Y. and Lee, T. (2010). The Effects of Scaffolding-Based Courseware for the Scratch Programming Learning on Student Problem Solving Skill. In *Proceedings of the 18th International Conference on Computers in Education*, *(pp. 723-727)*. Putrajaya, Malaysia.

Norwawi, N. M., Abdulsalam, S. F., Hibadullah, C. F. and Shuaibu, B. M. (2009). Classification Of Students' Performance in Computer Programming Course According to Learning Style. *Data Mining and Optimization.* Kajand, Malaysia.

Paechter, M., Maier, B., and Macher, D. (2010). Students' expectations of and experiences in e learning: Their relation to learning achievements and course satisfaction. *Computers & Education, 54(1),* 222-229.

Paris,W. (2001). The role of self-regulated learning in contextual teaching: principles and practices for teacher preparation. Retrieved 11 April 2018, from https://files.eric.ed.gov/fulltext/ED479905.pdf

Pintrich , P. R., Smith, D.A., Garcia, T.,and McKeachie, W. J. (1991). A manual for the use of the motivated strategies for learning questionnaire (MSLQ). University of Michigan, Ann Arbor.

Pintrich, P. R.(2004). A conceptual framework for assessing motivation and self-regulated learning in college students. *Educational Psychology Review, 16(4),* 385-407

Pintrich, P. R., De Groot, E. V. (1990). Motivational and self-regulated learning components of classroom academic performance. *Journal of Educational Pschology, 82,* 33-40.

Pintrich, P.R. (2000). A motivational science perspective on the role of student motivation in learning and teaching contexts. *Journal of Educational Psychology, 95,* 667-686.

Puzziferro, M. (2008). Online technologies self-efficacy and self-regulated learning as predictors of final grade and satisfaction in college-level online courses. *American Journal of Distance Education, 22(2),* 72-89.

Ramalingam, V., LaBelle, D. and Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program, In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, June 28-30, 2004*, Leeds, United Kingdom.

Saeli, M., Perrenet, J., Jochems, W. and Zwaneveld, B. (2011). Teaching programming in secondary school: a pedagogical content knowledge perspective. *Informatics in Education, 10(1).*

Safari, E., Hejazi, M. (2017). Learning styles and self-regulation: an associational study on high school students in iran. *Mediterranean Journal of Social Sciences, 8*(1), 463-469.

Schunk, D.H. & Zimmerman, B. J. (Eds.). (1998). Self-regulated learning: From teaching to self reflective practice. New York: Guilford Press.

Shannon, S.V. (2008). Using metacognitive strategies and learning styles to create self-directed learners. *Institute for Learning Styles Journal, 1*,14-28.

Wang, C., Shannon, D., and Ross, M. (2013). Students' Characteristics, Self-Regulated Learning, Technology, Self-Efficacy, and Course Outcomes in Online Learning. *Distance Education, 34(3),* 302-323.

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program, *In Proceedings of The First International Workshop on Computing Education Research, 13-24,* Seattle, WA, USA.

Wiedenbeck, S., LaBelle, D. and Kain, V. (2004). Factors affecting course outcomes in introductory programming. *In Proceedings of the Sixteenth Annual Workshop of the Psychology of Programming Interest Group (PP*IG' 04) (pp. 97-109).

Zhang, L., Jia, J., Wang, B., Amanai, K., Wharton, K. A., & Jiang, J. (2006). Regulation of wingless signaling by the CKI family in Drosophila limb development. *Developmental Biology*, 299(1), 221-237.

Zimmerman, B. J. (2008). Investigating self-regulation and motivation: Historical, background, methodological developments, and future prospects. *American Educational Research Journal*, 45, 166−183.

Zimmerman, B. J., & Martinez-Pons, M. (1990). Student differences in self-regulated learning: Relating grade, sex, and giftedness to selfefficacy and strategy use. *Journal of Educational Psychology, 82,* 51-59.

Zimmerman, B.J. (2002). Becoming a Self-Regulated Learner: An Overview, *Theory Into Practice, 41(2),* 64-70,

Zywno, M. S. (2003). A contribution to validation of score meaning for Felder-Soloman's index of learning styles. *In Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition* (Vol. 119, pp. 1-5). Washington, DC: American Society for Engineering Education.

Using Grey-based Mathematical Equations of Decision-making as Teaching Scaffolds: from an Unplugged Computational Thinking Activity to Computer Programming

Meng-Leong How[1]
Chee-Kit Looi[1]

[1]National Institute of Education, Nanyang Technological University, Singapore

DOI: 10.21585/ijcses.v2i2.24

Abstract

Computational Thinking (CT) is pervasive in our daily lives and is useful for problem-solving. Decision-making is a crucial part of problem-solving. In the extant literature, problem-solving strategies in educational settings are often conveniently attributed to intuition; however, it is well documented that computer programmers might even have difficulty describing about their intuitive insights during problem-solving using natural language (such as English), and subsequently convert what has been described using words into software code. Hence, a more analytical approach using mathematical equations and descriptions of CT is offered in this paper as a potential form of rudimentary scaffolding, which might be useful to facilitators and learners of CT-related activities. In the present paper, the decision-making processes during an unplugged CT activity are delineated via Grey-based mathematical equations, which is useful for informing educators who may wish to explain to their learners about the various aspects of CT which are involved in the unplugged activity and simultaneously use these mathematical equations as scaffolds between the unplugged activity and computer code programming. This theoretical manuscript may serve as a base for learners, should the facilitator ask them to embark on a software programming activity that is closely associated to the unplugged CT activity.

Keywords

grey-based mathematical equations, decision making, computational thinking, scaffolding for teaching, computer software programming, unplugged computational thinking activity

1. Introduction

In computer programming education, there might be an overemphasis on students' acquisition of the syntax of a programming language; often at the expense of development of problem-solving skills (McGill & Volet, 1997). Somers (2017) notices that programmers do not work on a problem directly. He quoted Nancy Leveson, a professor at the Massachusetts Institute of Technology who has been studying software safety for 35 years, who explains, "The problem is that software engineers don't understand the problem they're trying to solve, and don't care to. The reason is that they're too wrapped up in getting their code to work. The serious problems that have happened with software have to do with requirements, not coding errors." Hence, development of problem-solving skills should be a high priority for computer programming education.

Griffin (2016) points out that it is important for computer programmers to develop a mental model of a notional machine (du Boulay, O'Shea, & Monk, 1981), which is a rudimentary model that describes the instructions of a computer program for problem-solving. Strong interest in how the computer programmer could

develop this mental model (by researchers such as Grover, Pea, & Cooper, 2015; Hu, 2011; Selby, 2013; Wing, 2008), have more precisely explicated this mental model of a notional machine into what is now known as Computational Thinking (CT). A generally accepted definition of CT is still developing (Selby, 2013); even the very definition of individual constituents of CT such as the concept of abstraction is still evolving (Cetin & Dubinsky, 2017). Nevertheless, in the present paper, for the purpose of "operationalising" CT concepts for utilisation of mathematical equations in decision-making and problem-solving using a computer programming language, we follow the conceptual framework offered by Gouws et al. (2013) who have more concisely elucidated CT concepts specifically for the field of education. The constituents of this mental model of CT offered by Gouws et al. (2013) include decomposition, algorithmic thinking, abstraction of data and functionality, evaluation, and generalisation. Decomposition refers to the process of breaking down a problem into multiple steps, in order to solve it. Algorithmic thinking refers to the repetitive execution of patterns of instructions, which might involve loops for iteration or recursion. Abstraction of data and functionality refers to the notion of representations in data storage and the manipulation of those data in functions. Generalisation refers to the ability to create adaptable solutions that are reusable for a wider range of problems. Evaluation is the ability to select the best solution for a given problem, as well as to identify and correct errors.

CT is pervasive (Bundy, 2007); in our daily tasks, CT is particularly useful for problem-solving (Barr, Harrison, & Conery, 2011). Indeed, CT is indispensable to problem-solving in the real world, and is also considered to be essential in education (Wing, 2008). Efforts have already been made in many studies to delineate which aspects of CT might be explicitly learnt by a person who has participated in screen-based activities (by researchers such as Grover, 2015; Israel, Pearson, Tapia, Wherfel, & Reese, 2015; Monteiro, Salgado, Mota, Sampaio, & de Souza, 2016; Selby & Cynthia, 2015).

Visualisation of code can also be in the form of physical or kinaesthetic activities (also referred to as unplugged activities); not just on screen-based devices. Zagami (2012) posits that the computer programmer could understand programming concepts better from visualisation of how code works. Research into non-screen based unplugged activities (by researchers such as Bell et al., 2009; Cortina, 2015; Paul Curzon et al., 2014; Feaster, Segars, Wahba, & Hallstrom, 2011; Rodriguez, 2015; Taub, Armoni, & Ben-Ari, 2012; Taub, Ben-Ari, & Armoni, 2009; Thies & Vahrenhold, 2012; Thies & Vahrenhold, 2013) have demonstrated that they might potentially help learners to understand computing concepts kinaesthetically as they solve problems in the real world.

Research Problem

In the extant literature, decision-making strategies in problem-solving used by learners in educational settings might often just be conveniently assumed by educators to be naturalistic (Zsambok, 2014), or simply intuitive (Metcalfe & Wiebe, 1987; Pretz, 2008). However, it is well documented that computer programmers might have difficulties in describing about their intuitive insights during problem-solving using natural language (such as English), and subsequently convert what has been described using words into software code. In a previous study by Kordaki, Miatidis, and Kapsampelis (2008), the students constructed an algorithm intuitively in an activity using coins, but most of them had problems describing the procedure which they had used, when they tried to express it in natural language (English) and pseudocode. Boticki, Barisic, Martin, and Drljevic (2013) also observed that students might have difficulties translating their thoughts into a form that could be used in computers.

Decision making is that thinking which results in the choice among alternative courses of action;

problem solving is that thinking which results in the solution of problems (Taylor, 2013, p. 48). Further, Taylor (2013) also points out that the processes in decision-making are also important to problem-solving (p. 48). In the extant literature, besides the seminal work into computational models of decision-making done by Busemeyer and Johnson (2004), far too little attention has been paid to decision-making in problem-solving skills for computer programming education, and in particular for CT.

Wing (2008) proffers that CT is a form of analytical thinking which shares with mathematical thinking, engineering thinking, and scientific thinking in similar ways in which we might approach the understanding and modelling of real world phenomena, in order to solve problems. She points out that many sciences and engineering disciplines also rely on simulations of mathematical models of physical processes found in nature. Mathematical modelling has also been utilised in the field of education by educators, students, and researchers (Stillman, Blum, & Biembengut, 2015). In education, mathematical modelling has been employed as a strategy for building up systems of knowledge (D'Ambrosio, 2015), for students who do not solve problems independently to share and refine mathematical models through dual modelling teaching (Kawakami, Saeki, & Matsuzaki, 2015), for exploring interconnections between real-world and application tasks (Ng & Stillman, 2015), and as visualisation tactics for solving real-world tasks (Brown, 2015).

The authors of the present paper do concede that using mathematical equations to model a phenomena and subsequently converting those mathematical equations into computer programming code is nothing new; in fact, it is a part of Computational Science (Humphreys, 2004). In computational science, computational models are usually presented as mathematical models because they can be analysed and even run as simulations using computers to better understand the characteristics of the phenomenon being studied. For example, computational fluid dynamics (Chung, 2010) refers to the computational modelling of fluid dynamics. Computational finance (Ugur, 2008) refers to the computational modelling of financial-related systems. Computational biology (Waterman, 1995) refers to the computational modelling of biological-related systems. In the same token, computational thinking (Wing, 2006) could be construed as the computational modelling of thinking. Laudable efforts have been made by computational thinking researchers (such as Lu & Fletcher, 2009; Weintrop et al., 2016) to illustrate key concepts in computational thinking which involved the use of mathematical equations; however, these studies seem to be solely focused on which aspects of computational thinking were involved when a person encounters a mathematical formula or algorithm. Currently, there is a dearth of computational models about the "thinking" part of computational thinking in the extant literature. The present paper purports to explore this "thinking" part of computational thinking via the decision-making portions of problem-solving; first from the perspective of a human learner playing with the programmable toy mouse in an unplugged computational thinking activity, and subsequently from the perspective of a computer programmer who is programming the software version of a self-navigating mouse that can autonomously reach its objectives.

Since CT purports to enable humans to analyse problems, and to communicate the corresponding solutions using computational terms that are ultimately meant for computers to comprehend and execute, it follows that mathematical modelling might be well suited for understanding the decision-making processes involved during an unplugged CT activity. The mathematical model can also be utilised as a valuable resource should the facilitator choose to ask the learners to implement a software programme to describe the decision-making strategies that might be involved.

In the present paper, the decision-making processes during an unplugged activity, which utilises a programmable toy mouse as a simple example, are depicted via mathematical equations, to elucidate which aspects of CT might be involved in decision-making during problem solving. The analytical approach of using

mathematical equations and descriptions of CT is offered in this paper as a potential form of rudimentary scaffolding between CT concepts and software programming, which might be useful to facilitators and learners of CT-related activities.

The rest of this paper is organised as follows: in the next section, the basic building blocks for the mathematical modelling of decision-making during the unplugged CT activity will be presented. Using the conjectures from these mathematical equations, the aspects of CT involved, together with some examples of Python programming code that correspond to the mathematical equations, will be presented in the discussion section. In the present paper's hypothetical scenario, the programmer needs to implement parts of the Python code to create a software-based self-navigating mouse that can avoid obstacles and autonomously move towards its goals. Finally, the direction of future research will be presented in the conclusion section.

2. Mathematical modelling of decision-making in unplugged ct activity

In the context of this research, a simple case of an unplugged CT activity which involves a programmable toy mouse (see Figure 1) will be used. In this unplugged CT activity, the learner operating the programmable mouse must evaluate the possible consequences when trying to achieve the pre-determined objectives. Since computational modelling usually involves the use of mathematical models, it follows that using mathematical models to depict decision-making in problem-solving might allow us to clearly see the aspects of computational thinking involved. A grey-based approach (Deng, 1989; Liu & Lin, 2010; Liu, Yang, & Forrest, 2016) of mathematical modelling is utilised in this paper, because it excels in modelling phenomena in situations where there might be uncertainty, scarcity of quantitative data, or incomplete information; situations which learners often find themselves in during problem-solving exercises. Further, although the entire maze could be easily observed by a human, a software-based self-navigating mouse that the programmer is trying to create – having sensors only on its front, left, right, and rear – can only detect whether there is any object in its immediate vicinity. It does not have a "bird's eye view" of the entire maze. Also, the programmer might need to implement the features to enable the software-based mouse to "decide" whether an object is an obstacle or a goal it is trying to reach. It might also need to learn from its previous attempts and "predict" the next step that it needs to take. Accordingly, we use the conceptual notion of "black" to indicate completely unknown information, "white" to indicate completely known information, and "grey" to indicate partially known and partially unknown information (Liu & Lin, 2010, p. 15). Grey-based mathematical equations are used because they could potentially be used to address issues of uncertainty that might be countered when a programmer is trying to implement the self-navigating features of the software-based mouse in computer code.

Figure 1: Unplugged activity which uses a programmable toy mouse

The basic building blocks for modelling decision-making during problem-solving, which is adapted from the Grey Models of Decision Making, first developed by Liu & Lin (2010, p. 197) into the context of the programmable toy mouse in this unplugged CT activity, is presented as follows:

Let the four key elements of decision-making be events, countermeasures, effects, and objectives. Let the totality of all events which might be encountered by the programmable toy mouse in the unplugged activity be denoted as

$$E = \{e_1, e_2, \ldots e_n\} \quad (1)$$

where e_i represents the ith event within the set of events, where $i = 1, 2, 3, \ldots, n$, such that the event e_1 precedes e_2 and e_2 precedes e_3 and so forth.

A countermeasure (Oxford Living Dictionaries, 2017) is an actionable process or choice that could be taken (or conversely, not taken) to mitigate the effects of an event. The term "countermeasure" is used instead of "action" because of the implication that it could be taken or not taken after considerations during the decision-making process. The totality of all countermeasures is defined as the set of countermeasures, denoted as

$$C = \{c_1, c_2, \ldots, c_m\} \quad (2)$$

and c_j represents the jth countermeasure within the set of countermeasures, where $j = 1, 2, \ldots m$. Actions taken by the programmable mouse (see Figure 2) can be considered as countermeasures, with the action "forward" denoted as c_1, "rotate left" denoted as c_2, "rotate right" denoted as c_3, and "reverse" denoted as c_4. Thus, the set of countermeasures for the programmable toy mouse is denoted as

$$C = \{ c_1, c_2, c_3, c_4 \} \quad (3)$$

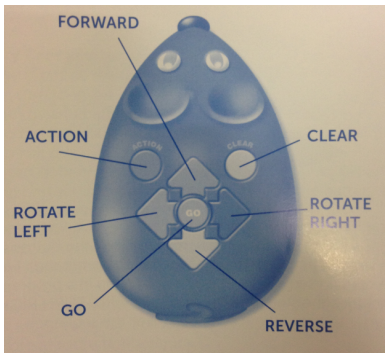

Figure 2: Buttons on the programmable toy mouse

The set of decision schemes $S = E \times C$ can be represented by the Cartesian product

$$E \text{ x } C = \{ (e_i, c_j) \mid e_i \in E, c_j \in C \} \quad (4)$$

of the set of events E, and the set of countermeasures C, where each pair of decision scheme $s_{ij} = (e_i, c_j)$, for any $e_i \in E$, $c_j \in C$. Hence, a set of decision schemes for the programmable toy mouse, which represents the totality of all the various combinations of moves it can make, is denoted as

$$S = E \times C = S_{ij} = (e_i, c_j) = \{ S_{11}, S_{12}, \ldots, S_{14}, S_{21}, \ldots, S_{24}, S_{31}, \ldots, S_{34}, \ldots \} \quad (5)$$

A set of decision schemes not only can be used to represent the totality of the various ways the programmable toy mouse can move that an individual learner has considered, if there is only one learner.

Especially noteworthy is, it can also be used to represent the totality of the various ways of solving the problem that a group of learners has discussed about together, if there is more than one learner involved, such in this instance, where many learners are involved in the decision-making process to discuss how best to move the programmable toy mouse.

In this unplugged activity, the facilitator can first explain to the learners that at each step, the programmer must decide whether to make the mouse move forward, or to rotate left or right, or to move in reverse. In our mathematical model, however, each "step" that the mouse takes can be technically considered to be an event. For example (see Figure 3), each step shall be technically referred to as an event in the computational model, and in each step, there is a corresponding countermeasure, which can be Forward, or Rotate Left, or Rotate Right.

For example, at event e_1, the programmable toy mouse might not be blocked by any obstacle in front, behind, or on its sides, so the programmer can choose to deploy countermeasure c_1 to move the mouse one square forward. At event e_2, the programmable mouse might encounter an event where the goal (the cheese) is located to its left, so the programmer needs to deploy the countermeasure c_2 to rotate left, and so forth.

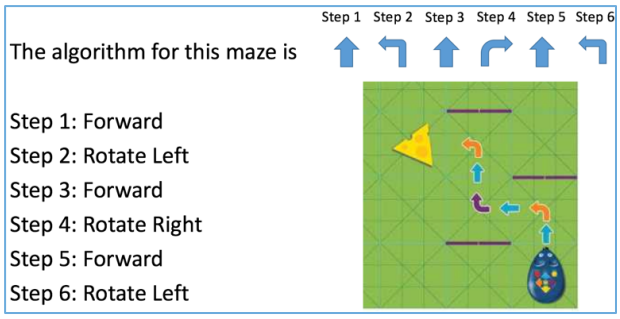

Figure 3: Example of a path utilised by a learner for the programmable toy mouse

Effect values of decision schemes

To encourage the learners to participate in this unplugged activity, the facilitator might like to consider explaining to them that there will be a competition where each team is required to discuss about the best steps for the mouse to take, before using the mouse. The rules of the competition can be presented to them in a slide (see sample slide in Figure 4).

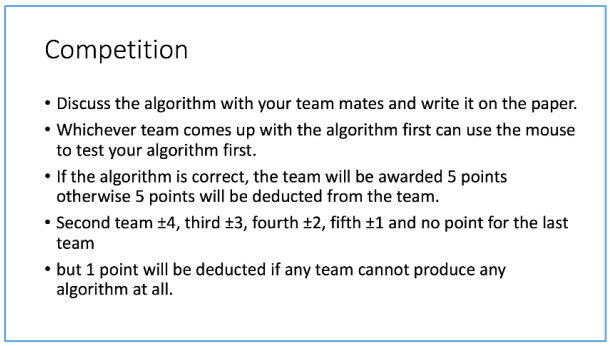

Figure 4: Rules of the competition for the teams of learners participating in the unplugged CT activity

Effect value of decision scheme

Let a set of decision schemes be denoted as

$$S = \{ S_{ij} = (e_i, c_j) \mid e_i \in E, c_j \in C \} \quad (6)$$

Further, let $u_{ij}^{(k)}$ represent the effect value of the set of decision scheme S_{ij} with respect to the objective k, where R represents the set of all real numbers. If we suppose that $u_{ij}^{(k)}: S \longmapsto R$, that is, the effect value $u_{ij}^{(k)}$ can be mapped from the set of decision schemes to a set of real numbers with respect to the objective k, it follows that a particular decision scheme s_{ij} can also be mapped to an effect value, which can be denoted as $s_{ij} \longmapsto u_{ij}^{(k)}$. For example, in the context of the points award system of the competition in this unplugged CT activity (see Figure 4), the learners in each team are encouraged to discuss among themselves to propose a solution (a particular decision scheme which is selected from a set of decision schemes) for their team and try it on the programmable toy mouse. Hence, the effect values of the teams' decision schemes can be directly manifested in the real values of the points that they score (or lose).

Equivalent countermeasures

If two different actions (also referred to as countermeasures) of the programmable toy mouse can contribute to achieving the same objective, we can denote it as follows: if the countermeasures c_j and c_h are equivalent with respect to objective k, it can be denoted as $c_j \cong c_h$. Hence, the set with equivalence class of countermeasure c_h to the event e_i with respect to objective k can be denoted as

$$C_i^{(k)} = \{\, c \mid c \in C, c \cong c_h \,\} \qquad (7)$$

Suppose the two effect values $u_{ij}^{(k)}$ and $u_{ih}^{(k)}$ are equivalent, then this effect equivalence can be denoted as $u_{ij}^{(k)} = u_{ih}^{(k)}$. For example (see Figure 5), the effect value in Decision Scheme A can be considered to be equivalent to the effect value in Decision Scheme B, because the mouse takes the same number of countermeasures to reach the cheese.

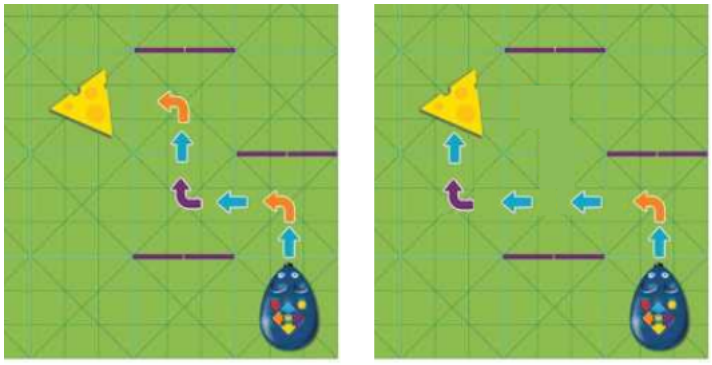

Figure 5: Example of equivalent effect values

Superiority of a countermeasure

If a countermeasure is considered to be better than another countermeasure for the programmable toy mouse, it can be denoted as follows: if the effect value $u_{ij}^{(k)}$ is greater than the effect value $u_{ih}^{(k)}$, it can be denoted as $u_{ij}^{(k)} > u_{ih}^{(k)}$. If the countermeasure c_j is superior to c_h in response to event e_i with respect to

objective *k*, it can be denoted as $c_j \succ c_h$. Hence, it follows that the superior set of countermeasures c_h to the event e_i with respect to objective *k* can be denoted as

$$C_{ih}^{(k)} = \{ c \mid c \in C, c \succ c_h \} \quad (8)$$

Superiority of a decision scheme

After discussion amongst the learners in their respective teams, if a set of emergent decision scheme is considered to be superior by the team members, it can be expressed as follows: if the effect value $u_{ij}^{(k)}$ must be greater than the effect value $u_{ih}^{(k)}$ to achieve the objective, it can be denoted as $u_{ij}^{(k)} > u_{ih}^{(k)}$. If the decision scheme s_{ij} is superior to s_{hl} with respect to objective *k*, it can be denoted as $s_{ij} \succ s_{hl}$ and hence the set of superior decision scheme can be denoted as

$$S_{hl}^{(k)} = \{ s \mid s \in S, s \succ s_{hl} \} \quad (9)$$

For example (see Figure 6), if decision scheme A is contributed by Learner A, decision scheme B is contributed by Learner B, and decision scheme C is contributed by Learner C, and suppose in the group discussion, the three learners come to a decision that decision scheme A is superior because the programmable toy mouse would be able to reach the cheese if this is chosen, we can say that decision scheme A is superior compared to decision schemes B and C.

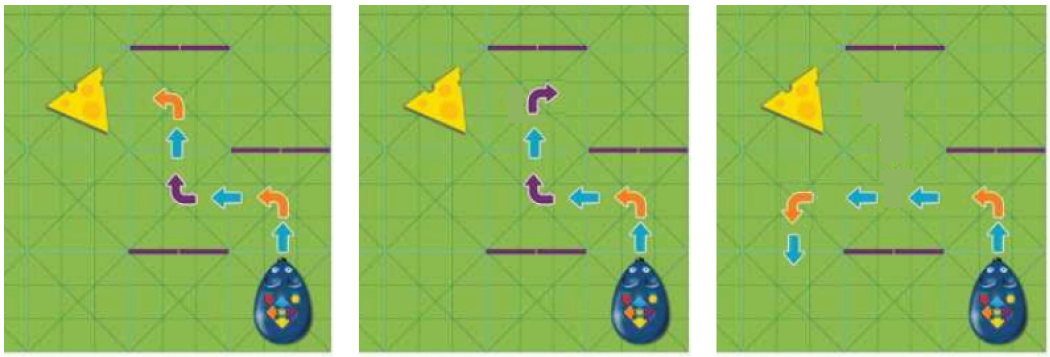

Figure 6: Example of a superior decision scheme

Threshold values of decision effects

In the competition within this unplugged activity, the teams of learners are motivated to discuss and present their perceived solution quickly, so that they can score higher points; however, if their solution of the algorithm is incorrect, they might be penalised too, so this can be expressed as follows: let $d_1^{(k)}$ be the upper threshold value (the points that the team can score if its solution is correct), and $d_2^{(k)}$ be the lower threshold value (the points that the team can score if its solution is incorrect) of the decision scheme s_{ij} with respect to the single objective *k*, and *r* be the value between the range of $d_1^{(k)}$ and $_2^{(k)}$. It follows then that the one-dimensional grey target for objective *k* can be denoted as

$$S^1 = \{ r \mid d_1^{(k)} \leq r \leq d_2^{(k)} \} \quad (10)$$

and a satisfactory effect value with respective to objective *k* can be denoted as

$$u_{ij}^{(k)} \in [d_1^{(k)}, d_2^{(k)}] \quad (11)$$

Decision-making with multiple objectives

Suppose $u_{ij}^{(k)}$ represents the effect value of decision scheme s_{ij} with respect to a single objective *k*. If s_{ij} is a feasible decision scheme which can contribute to achieving the objective *k*, it can be denoted as $s_{ij} \in S^1$. This applies to situations which involve a single objective.

For grey targets of decision-making with multiple objectives, if there are two objectives for instance (see Figure 7), we can assume that $d_1^{(1)}$ and $d_2^{(1)}$ to be the lower and upper threshold values of the decision effects of objective 1, where $r^{(1)}$ represents the value between the range of $d_1^{(1)}$ and $d_2^{(1)}$. We can also assume $d_1^{(2)}$and $d_2^{(2)}$ to be the lower and upper threshold values of the decision effects of objective 2 , where $r^{(2)}$ represents the value between the range of $d_1^{(2)}$ and $d_2^{(2)}$. Hence, the grey target of two-dimensional decision-making can be denoted as

$$S^2 = \{ r^{(1)}, r^{(2)} \mid d_1^{(1)} \le r^{(1)} \le d_2^{(1)}, d_1^{(2)} \le r^{(2)} \le d_2^{(2)} \} \qquad (12)$$

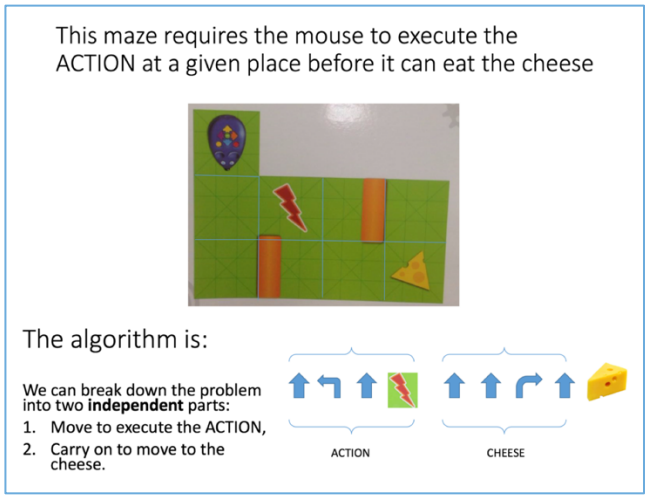

Figure 7: Multiple objectives of the programmable toy mouse

If this effect vector of s_{ij} satisfies the effect value u_{ij} such that $u_{ij} = \{u_{ij}^{(1)}, u_{ij}^{(2)}\} \in S^2$, then s_{ij} can be considered to be a superior decision scheme with respect to objectives 1 and 2. It also follows that c_j can be considered to be a superior countermeasure for event e_i with respect to objectives 1 and 2.

Suppose $d_1^{(1)}$, $d_2^{(1)}$, $d_1^{(2)}$, $d_2^{(2)}$, … where $d_1^{(s)}$ and $d_2^{(s)}$ represent the lower and upper threshold values of decision effects with respect to objectives 1, 2, … , *s*. A grey-target with a *s*-dimensional decision-making scheme can be denoted in Euclidean space as

$$S^s = \{ r^{(1)}, r^{(2)}, \dots, r^{(s)} \mid d_1^{(1)} \le r^{(1)} \le d_2^{(1)}, d_1^{(2)} \le r^{(2)} \le d_2^{(2)}, \dots, d_1^{(s)} \le r^{(s)} \le d_2^{(s)} \} \qquad (13)$$

Hence, if s_{ij} is a superior decision scheme, where $u_{ij}^{(k)}$ represents the effect value of the decision scheme s_{ij} with respect to the objective *k*, and *k* = 1, 2, …, *s*, then effect vector can be denoted as

$$u_{ij} = \{u_{ij}^{(1)}, u_{ij}^{(2)}, \dots, u_{ij}^{(s)}\} \in S^s \qquad (14)$$

These grey-targets of decision-making represent the locus of superior effects. In reality, however, it

might be almost impossible to achieve absolute optimization of an outcome. Nevertheless, in analysis we endeavour to strive for the quasi-optimal outcome, where the decision scheme and its corresponding countermeasures are quasi-optimal; which is to say, they are the best choices among the available decision schemes and their corresponding countermeasures. As a cautionary note, however, the quasi-optimal solution presented by a team of learners might not always be the correct solution; in fact, it could even be incorrect.

Time Series: Memories of Sets of Decision Schemes

So far, the discussion has focused only on static decision schemes with a fixed moment in time. Grey-based decision-making can also focus on changes of the decision effect over time (Liu et al., 2016). Let us suppose that in a hypothetical scenario, the facilitator might wish to consider asking the learners to develop a software-based self-navigating mouse that can perform autonomous problem-solving in a series of different maze challenges, not just in one maze challenge. Instead of static decision schemes, the concept of time can now be included; as time advances forward, the changing decision effects can also be considered.

Memory plays an important role in problem solving (Reber & Kotovsky, 1997). In the software-based self-navigating mouse's multiple attempts at problem-solving, which involves the notion of time, it has to "remember" the consequential effects of its previous attempts, before another attempt is made to solve a similar problem in the future. As such, the facilitator might also consider asking the learners to implement a rudimentary type of memory into the software-based self-navigating mouse, so that it can "remember" its previous moves in the form of a time series. Suppose a set of events is represented by $E = \{ e_1, e_2, e_3, \ldots, e_n \}$, a set of countermeasures is represented as $C = \{ c_1, c_2, \ldots, c_m \}$, and the set of decision schemes is represented by $S = \{ s_{ij} = (e_i, c_j) \mid e_i \in E, c_j \in C \}$, then it follows that the time series of decision effect of the decision scheme s_{ij} with respect to the objective k can be denoted as

$$u_{ij}^{(k)} = (u_{ij}^{(k)}(1), u_{ij}^{(k)}(2), \ldots, u_{ij}^{(k)}(h)) \qquad (15)$$

This section has described some mathematical equations that might be used to depict the decision-making processes that might be involved in the programmable toy mouse in the unplugged CT activity. The next section will present some of the CT concepts and their corresponding Python programming code that might be useful as scaffolds for the teacher in the facilitation of a discussion with the students about CT concepts in decision-making and problem-solving, and how to implement them using computer programming code.

3. Discussion

So far, these grey-based mathematical equations have tried to depict the decision-making processes that might be involved in the unplugged CT activity to educe (meaning: to draw out) the problem-solving abilities of the learners. Sometimes, after an unplugged CT activity has been conducted, the facilitator might wish to continue with a code writing activity for the learners, if they already have experience in software programming. For example, the facilitator might consider asking the learners to write code to implement the mouse in software, in such a way so that it has autonomous decision-making abilities. Besides text syntax-based programming languages such as Python, C++, and Java, software programmes can also be developed using mathematical equations. Currently, a software which utilises symbolic computing and can accept mathematical symbols as part of its programming syntax to create simulations is Mathematica (Wolfram Research Incorporated, 2017). Should the facilitator choose to explicitly explain the CT concepts involved in the unplugged CT activity to the learners, so that by analogous association, they can programme decision-making capabilities in the software-based

autonomous self-navigating mouse, the following information might be useful for the programmers, regardless of the choice of programming language to be used. In the current section, some suggested snippets of computer programming will be offered as illustrations to show how the afore-mentioned mathematical equations of decision-making can be used as scaffolds by the teacher for possible discussions of computer programming with the students. The Python programming language is used in the present paper, because it has become quite popular for learning programming in schools. It had also gained traction as a programming language for development work in machine learning, deep learning, and artificial intelligence. In the present paper, the Python code snippets are not intended to be complete solutions; they merely serve as scaffolds for the teacher to discuss about programming concepts and CT concepts with the students. This section purports to make explicit the CT concepts for implementing decision-making capabilities in the software-based self-navigating mouse.

Computational Thinking in Decision-making: Abstraction

The CT concept of abstraction of data can be applied to the events (see Table 1) which can be represented as lists or arrays in the software-based self-navigating autonomous mouse. For the purpose of keeping this example simple, the number of events is assumed to be 5. It is also assumed that in this game, the self-navigating mouse is not allowed to reverse. The values inside each event are assumed to be some data "sensed" by the software version of the self-navigating mouse, perhaps via machine-vision or via proximity sensors. Let us assume that this software-based self-navigating mouse has three sensors, one on its front, one on its left side, and one on its right side. In each event that the mouse encounters, the event e_i can be represented as an array with 3 values, each from its front, left, and right sensor respectively. For example, if there is a cheese in front of the mouse, its value would be: 2, if there is an obstacle on its left, its value would be: -1, if there is no object on its right, its value would be: 0. Hence, it can be represented as an array in Python code e_1 = [2, -1, 0]. Therefore, if it encounters 5 events, it can be represented in Python code as follows:

Table 1: Mathematical equation of an array of events and its corresponding Python code

From Equation 1: A set of events	*Example of corresponding Python code*
$E = \{e_1, e_2, \ldots e_5\}$	`# create array` `from numpy import array` `# Each array of Event e1 to e5 filled with` `# data from Front, Left & Right sensors` `e1 = [0,-1,0]; e2 = [0,1,0]; e3 = [0,0,0]` `e4 = [-1,0,0]; e5 = [0,2,0]` `# create array of a Set of Events` `E = [e1, e2, e3, e4, e5]` `a = array(E)` `# display array` `print(a)`

The CT concept of abstraction of data can be applied to the events (see Table 2) by representing them as an immutable tuple or alternatively as an array in the software-based self-navigating autonomous mouse. For the purpose of keeping this example simple, the number of events is assumed to be 4, where actions taken by the self-navigating autonomous mouse can be considered to be countermeasures, with the action "forward" denoted as c_1, "rotate left" denoted as c_2, "rotate right" denoted as c_3 and "reverse" denoted as c_4. For ease of computation by the software-based self-navigating autonomous mouse, the value of c_1 is 1, the value of c_2 is 2, the value of c_3 is 3, and the value of c_4 is 4.

Table 2: Mathematical equation of an array of countermeasures and its corresponding Python code

From Equation 3:	*Example of corresponding Python code*
A set of 4 countermeasures $C = \{ c_1, c_2, c_3, c_4 \}$	`# create array` `from numpy import array` `# declare variables c1 to c4` `# and initialize them with values` `# 1=Front; 2=Left; 3=Right; 4=Reverse` `c1 = 1; c2 = 2; c3 = 3; c4 = 4` `# create a tuple of Countermeasures` `# as we want the values to be immutable` `C = {c1, c2, c3, c4}` `# convert the tuple C into an array` `Countermeasures = array(C)` `# display array` `print(Countermeasures)`

Besides representing data as variables in lists or arrays, data can also be presented in the form of a Cartesian structure in Euclidean space during the decision-making process (see Equation 5) in the software-based self-navigating autonomous mouse. This Cartesian structure can be easily created using Python; it involves the usage of vertically stackable arrays to create a multi-dimensional matrix (see Table 3). Multi-dimensional matrices are also referred to as tensors, which could be used to store data of different data-types "sensed" by the software-based self-navigating autonomous mouse from its sensors. In real-world practical applications, tensors – which are useful for storing massive amounts of digital data from pixels of images, audio data, spatial data, and so forth – are the cornerstone of data structures in artificial intelligence programming-related software such as TensorFlow, Theano, and Keras.

Table 3: Mathematical equation of a multi-dimensional matrix and its corresponding Python code

From Equation 5:	*Example of corresponding Python code*
An E by C dimension Cartesian structure formed from the equation which represents Events and Countermeasures. $E \times C = S_{ij}$ $= \{ S_{11}, S_{12}, \ldots, S_{14}, S_{21}, \ldots, S_{24}, S_{31}, \ldots, S_{34}, \ldots \}$	`# create array with vstack` `from numpy import array` `from numpy import vstack` `# initialize the variables` `e1 = [0,-1,0]; e2 = [0,1,0]; e3 = [0,0,0]` `e4 = [-1,0,0]; e5 = [0,2,0]` `c1=1; c2=2; c3=3; c4=4` `# create first array` `E = array([e1,e2,e3,e4,e5])` `print(E)` `# create second array` `C = array([c1,c2,c3])` `print(C)` `# vertical stack` `S = vstack((E, C))` `print(S)`

Computational Thinking in Decision-making: Evaluation

The CT concept of evaluation could potentially be applied in this manner: let us suppose that the software-based self-navigating mouse has autonomous decision-making capabilities to select the best quasi-optimum solution using the decision-making process to determine the superiority of a countermeasure (see Equation 8). The following is a simple contrived example of using Python code to calculate the magnitude of an array of possible countermeasures, so that a "superior" countermeasure can be determined:

Table 4: Python code to calculate the magnitude of an array to determine which direction the self-navigating mouse should take

From Equation 8: Superiority of a Countermeasure $C_{ih}^{(k)} = \{ c \mid c \in C, c \succ c_h \}$	*Example of corresponding Python code to calculate and compare the magnitudes of three arrays of events data collected by the self-navigating mouse's sensors on its front, left, and right, to determine which is the superior way to be taken by the self-navigating mouse; that is, the superior countermeasure (direction) that would allow that higher value to be manifested.* `from numpy import array` `# the arrays of Events e1, e2, e3 are each filled` `# by data from Front, Left, and Right sensors` `e1 = [0,-1,0]; e2 = [0,1,0]; e3 = [0,0,0]` `sum_e1 = sum(e1)` `sum_e2 = sum(e2)` `sum_e3 = sum(e3)` `#Output the magnitude of each array` `print ("Sum of e1: " +str(sum_e1))` `print ("Sum of e2: " +str(sum_e2))` `print ("Sum of e3: " +str(sum_e3))` `if ((e1 >= e2) & (e1 >= e3)):` `  print ("The mouse should go STRAIGHT")` `elif ((e2 >= e1) & (e2 >= e3)):` `  print ("The mouse should go LEFT")` `elif ((e3 >= e1) & (e2 >= e2)):` `  print ("The mouse should go RIGHT")`

The CT concept of evaluation can also be applied by the software programmer to determine the superiority of a decision scheme (see Equation 9). Usually, in a self-navigating autonomous machine, the evaluation is not determined by the human programmer, but by the machine itself using algorithms that are useful for machine vision, image pattern recognition, path finding, and so forth. In practical terms, the programmer might simply need to "feed" the data (most probably contained in the data structure of a matrix) to the machine learning or deep learning algorithm. Comparison of numerous multi-dimensional matrices (multiple decision schemes) can then be performed by the machine learning or deep learning algorithm inside the self-navigating autonomous mouse to determine the "superiority" of a decision scheme (one single matrix) in a set of decision schemes (numerous matrices).

Computational Thinking in Decision-making: Decomposition

The CT concept of decomposition could be applied in decision-making with multiple objectives (see Equation 12) in the software-based self-navigating mouse. For instance, if there are two objectives, and if the self-navigating mouse is required to reach the Action symbol, as well as the cheese symbol on the board, it

would be required to break down the steps to be taken to achieve those two objectives.

Computational Thinking in Decision-making: Algorithmic Thinking

In addition to the CT concept of decomposition in decision-making for multiple objectives, the CT concept of algorithmic thinking, in conjunction with the concept of utilising the threshold values of decision effects (see Equation 13) could be applied to the software-based self-navigating mouse so that it can "think" about using similar methods for reaching multiple objectives, even though the objectives may look different. For example, the same method that the software-based self-navigating autonomous mouse can use to reach the Action symbol, that is, by comparing countermeasures to determine which one is superior (for instance, one path which uses fewer steps), can also be applied to reach the cheese. In terms of practical application, the software programmer might wish to consider using a Time Series (see Equation 15) so that the events, countermeasures, decision schemes, and decision effects considered and taken (or considered but not taken) by the software-based self-navigating mouse can be "stored" for analysis to determine which next step to take (forward, left, or right).

Table 5: Python code to utilise a Time Series and make a one-step prediction

From Equation 15:	*Example of corresponding Python code which utilises the concept of Time Series to apply the CT concept of generalisation to analyse the pattern in the data, and subsequently make a one-step prediction*
Data stored in a Time Series $u_{ij}^{(k)} = (u_{ij}^{(k)}(1), u_{ij}^{(k)}(2), \ldots , u_{ij}^{(k)}(h))$	`# load Auto-regression model from file and make a one-step prediction` `from statsmodels.tsa.ar_model import ARResults` `import numpy` `# load model` `model = ARResults.load('model.pkl')` `data = numpy.load('data.npy')` `last_observation = numpy.load('observation.npy')` `# make prediction` `predictions = model.predict(start=len(data), end=len(data))` `# transform prediction` `predicted_value = predictions[0] + last_observation[0]` `print('Prediction: %f' % predicted_value)`

Computational Thinking in Decision-making: Generalisation

Finally, the CT concept of generalisation in decision-making might be implemented in the software-based self-navigating mouse in a manner that can combine the sets of events, countermeasures, decision schemes, and objective effects into a decision-making algorithm, so that they can be utilised in problem solving, for example, to autonomously predict the best route to take in new mazes. Practically, one of the ways might be for the programmer to consider implementing a LSTM (Long Short-Term Memory) recurrent neural network, which is a cornerstone of Machine Learning/Deep Learning for predicting new sequences (and in this context: paths) based on older data. The programmer may wish to consider implementing code to develop persistence (a form of memory) and perform analysis on the data from events, countermeasures, decision schemes, and decision effects, so that the self-navigating mouse can develop its own algorithmic thinking. More information about coding LSTM recurrent neural networks in Python can be perused at Brownlee's (2016) website.

4. Conclusion and future research

Grey-based mathematical equations have been utilised in the present paper to depict what might be involved in decision making during an unplugged CT activity. Mathematical modelling of decision-making might contribute to addressing a gap in the extant literature of CT research that has insofar not been studied much. An analytical approach using mathematical equations and descriptions of CT has been offered in this paper as a potential form of rudimentary scaffolding, which might be useful to facilitators and learners of CT-related activities. The mathematical equations of the decision-making processes posited in this theoretical manuscript may serve as a base for programmers, regardless of the programming language they prefer, should the facilitator wish to ask the learners to embark on a software programming activity that is closely associated to the unplugged CT activity.

Indeed, teachers/instructors might not need the mathematical equations in the present paper to teach an activity such as navigating in the maze. They might, however, find them to be useful as scaffoldings if software programming by the learners is involved after the conclusion of the unplugged CT activity. The hypothesis is that, if the teachers are exposed to a math model, they can be made aware of what the decision options are, and how to interpret the actions and results provided by students. Further, they might be more aware of the ramifications of the unplugged activity through its representation as a mathematical model. We hope future research can explore this hypothesis. Better still, if some instructors can create the model or fragments of the model, they can become even more conversant of the content knowledge to be taught and can build on the model to do the programming of the algorithm.

The existence of the problem-solving conceptual framework that has come to be referred to as computational thinking cannot be in doubt; however, what that structure is, might be another matter that is worthy of further research and exploration. As researchers seek to understand more about the various aspects of computing education, the utilisation of mathematical modelling might play a significant role in CT by, for example, describing it in more formal terms via mathematical equations to uncover aspects of CT that might be useful for programmers; should the need arise to implement them systematically in software code.

Acknowledgements

Support for this paper was provided by the project grant for: Researching and developing pedagogies using unplugged and computational thinking approaches for teaching computing in the schools (Project Number: OER 04/16 LCK)**.** Many thanks to Peter Seow, Longkai Wu, and Liu Liu for their help in designing and conducting the programmable toy mouse unplugged activity in a classroom.

References

Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking: A Digital Age Skill for Everyone. *Learning and Leading with Technology*, *38*(6), 20–23.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer Science Unplugged: School Students Doing Real Computing Without Computers. *Journal of Applied Computing and Information Technology*, *13*(1), 20–29.

Boticki, I., Barisic, A., Martin, S., & Drljevic, N. (2013). Teaching and learning computer science sorting algorithms with mobile devices: A case study. *Computer Applications in Engineering Education*, *21*, 41–50.

Brown, J. P. (2015). *Visualisation Tactics for Solving Real World Tasks*. (G. A. Stillman, W. Blum, & M. S. Biembengut, Eds.), *Mathematical Modelling in Education Research and Practice: Cultural, Social and*

Cognitive Influences.

Brownlee, J. (2016). Making Predictions with Sequences. Retrieved February 6, 2018, from https://machinelearningmastery.com/sequence-prediction/

Bundy, A. (2007). Computational Thinking is Pervasive. *Journal of Scientific and Practical Computing*, *1*(2), 67–69.

Busemeyer, J. R., & Johnson, J. G. (2004). Computational models of decision making. In *Blackwell handbook of judgment and decision making* (pp. 133–154).

Cetin, I., & Dubinsky, E. (2017). Reflective abstraction in computational thinking. *Journal of Mathematical Behavior*, *47*(November 2016), 70–80. https://doi.org/10.1016/j.jmathb.2017.06.004

Chung, T. J. (2010). *Computational fluid dynamics*. Cambridge university press.

Cortina, T. J. (2015). Broadening Participation: Reaching a broader population of students through "unplugged" activities. *Communications of the ACM*, *58*(3), 25–27. https://doi.org/10.1145/2723671

Curzon, P., McOwan, P. W. P., Plant, N., & Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 89–92. https://doi.org/10.1145/2670757.2670767

D'Ambrosio, U. (2015). Mathematical Modelling as a Strategy for Building-Up Systems of Knowledge in Different Cultural Environments. In G. A. Stillman, W. Blum, & M. S. Biembengut (Eds.), *Mathematical Modelling in Education Research and Practice: Cultural, Social and Cognitive Influences* (pp. 35–44).

Deng, J. (1989). Introduction to Grey System Theory. *The Journal of Grey System*, *1*, 1–24.

du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies*, *14*, 237–249.

Feaster, Y., Segars, L., Wahba, S., & Hallstrom, J. (2011). Teaching CS unplugged in the high school (with limited success). *ITiCSE*, 248–252. https://doi.org/10.1145/1999747.1999817

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in educational activities. *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '13*, 10. https://doi.org/10.1145/2462476.2466518

Griffin, J. M. (2016). Learning by Taking Apart: Deconstructing Code by Reading, Tracing, and Debugging. *Proceedings of the 17th Annual Conference on Information Technology Education (SIGITE '16)*, 148–153. https://doi.org/10.1145/2978192.2978231

Grover, S. (2015). "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. *Annual Meeting of the American Educational Research Association*, (650).

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Hu, C. (2011). Computational thinking. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, ITiCSE '11*, 223–227. https://doi.org/10.1145/1999747.1999811

Humphreys, P. (2004). *Extending ourselves: Computational science, empiricism, and scientific method*. Oxford University Press.

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers and Education*, *82*, 263–279. https://doi.org/10.1016/j.compedu.2014.11.022

Kawakami, T., Saeki, A., & Matsuzaki, A. (2015). *How Do Students Share and Refine Models Through Dual*

Modelling Teaching: The Case of Students Who Do Not Solve Independently. (G. A. Stillman, W. Blum, & M. S. Biembengut, Eds.), *Mathematical Modelling in Education Research and Practice: Cultural, Social and Cognitive Influences*.

Kordaki, M., Miatidis, M., & Kapsampelis, G. (2008). A computer environment for the learning of sorting algorithms: Design and pilot evaluation. *Computers & Education*, *51*, 708–723.

Liu, S., & Lin, Y. (2010). *Grey Systems: Theory and Applications*. Berlin: Springer-Verlag.

Liu, S., Yang, Y., & Forrest, J. (2016). *Grey Data Analysis*. Singapore: Springer-Verlag.

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking About Computational Thinking. In *SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer science education* (pp. 260–264). Chattanooga, TN, USA. https://doi.org/10.1145/1539024.1508959

McGill, T. J., & Volet, S. E. (1997). A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education*, *6504*(December), 37–41. https://doi.org/10.1080/08886504.1997.10782199

Metcalfe, J., & Wiebe, D. (1987). Intuition in insight and noninsight problem solving. *Memory & Cognition*, *15*(3), 238–246. https://doi.org/10.3758/BF03197722

Monteiro, I. T., Salgado, L. C. de C., Mota, M. P., Sampaio, A. L., & de Souza, C. S. (2016). Signifying software engineering to computational thinking learners with AgentSheets and PoliFacets. *Journal of Visual Languages and Computing*, (February 2016), 1–21. https://doi.org/10.1016/j.jvlc.2017.01.005

Ng, K. E. D., & Stillman, G. A. (2015). *Exploring Interconnections Between Real-World and Application Tasks: Case Study from Singapore*. (G. A. Stillman, W. Blum, & M. S. Biembengut, Eds.), *Mathematical Modelling in Education Research and Practice: Cultural, Social and Cognitive Influences*.

Oxford Living Dictionaries. (2017). Countermeasure. Retrieved December 4, 2017, from https://en.oxforddictionaries.com/definition/countermeasure

Pretz, J. E. (2008). Intuition versus analysis: Strategy and experience in complex everyday problem solving. *Memory & Cognition*, *36*(3), 554–566. https://doi.org/10.3758/MC.36.3.554

Reber, P., & Kotovsky, K. (1997). Implicit learning in problem solving: The role of working memory capacity. *Journal of Experimental Psychology: General*, *126*(2), 178.

Rodriguez, B. R. (2015). *Assessing Computational Thinking in Computer Science Unplugged Activities*. Colorado School of Mines. https://doi.org/10.1017/CBO9781107415324.004

Selby, C. (2013). Computational Thinking : The Developing Definition. *ITiCSE Conference 2013*, 5–8.

Selby, C. (2015). Relationships: Computational Thinking, Pedagogy of Programming, and Bloom's Taxonomy. *Proceedings of the Workshop in Primary and Secondary Computing Education*, 80–87. https://doi.org/10.1145/2818314.2818315

Somers, J. (2017). The Coming Software Apocalypse A small group of programmers wants to change how we code—before catastrophe strikes. Retrieved October 3, 2017, from https://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/

Stillman, G. A., Blum, W., & Biembengut, M. S. (Eds.). (2015). *Mathematical Modelling in Education Research and Practice: Cultural, Social and Cognitive Influences*.

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS Unplugged and Middle-School Students' Views, Attitudes, and Intentions Regarding CS. *ACM Transactions on Computing Education*, *12*(2), 1–29. https://doi.org/10.1145/2160547.2160551

Taub, R., Ben-Ari, M., & Armoni, M. (2009). The effect of CS unplugged on middle-school students' views of CS. *ACM SIGCSE Bulletin*, *41*(3), 99. https://doi.org/10.1145/1595496.1562912

Taylor, D. W. (2013). Decision making and problem solving. In *Handbook of organizations* (pp. 48–86).

Thies, R., & Vahrenhold, J. (2013). On plugging "unplugged" into CS classes. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE '13*, 365–370. https://doi.org/10.1145/2445196.2445303

Thies, R., & Vahrenhold, J. B. (2012). Reflections on outreach programs in CS classes: Learning objectives for "unplugged" activities. *SIGCSE12 Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 487–492. https://doi.org/10.1145/2157136.2157281

Ugur, Ö. (2008). *An introduction to computational finance*. World Scientific Books.

Waterman, M. S. (1995). *Introduction to computational biology: maps, sequences and genomes*. CRC Press.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London: Mathematical, Physical and Engineering Sciences*, (July), 3717–3725. https://doi.org/10.1109/IPDPS.2008.4536091

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33–35.

Wolfram Research Incorporated. (2017). Mathematica, Version 11.2, (2017). Champaign, IL.

Zagami, J. (2012). *Seeing is understanding: The effect of visualisation in understanding programming concepts*. Lulu.com.

Zsambok, C. E. (2014). *Naturalistic decision making*. Chicago: Psychology Press.

IJCSES

Volume 2, Issue 2

April 2018

ISSN 2513-8359