Informatics in Education, 2020, Vol. 19, No. 4, 543–568

DOI: 10.15388/infedu.2020.24

Teacher Perspectives on Introducing Programming Constructs through Coding Mobile-Based Games to Secondary School Students

Lara ATTARD, Leonard BUSUTTIL
Department of Technology and Entrepreneurship, Faculty of Education, Malta
e-mail: lcami02@um.edu.mt, leonard.busuttil@um.edu.mt

Received: June 2020

Abstract. Programming is one of the most important aspects of a Computing course. Teaching programming is a challenging task due to a number of factors, ranging from lack of student problem solving skills to different teaching methods. This paper focuses on Maltese Computing teachers' perspectives about the difficulties encountered when teaching programming to secondary school students in order to determine whether introducing programming to secondary school students through creating mobile-based games is an effective method to teach programming constructs. A resource pack consisting of various activities using MIT App Inventor 2 was created which incorporated constructivist approaches to teaching. This resource pack was reviewed by the teachers and their feedback was collected by means of a case study. The teachers agreed that developing mobile-based games would be highly stimulating to their students but there were uncertainties how this would affect students with different learning abilities and due to a general lack of computational thinking and problem-solving skills by most students.

Keywords: programming, computing, digital games, programming pedagogy.

1. Introduction

Programming is a skill which several students find difficult to grasp when studying Computing at secondary school level. Unfortunately, many students view this part of the syllabus as difficult to connect to. Traditionally, programming is perceived as difficult to master (Bennedsen, Caspersen, & Kölling, 2008). One of the main reasons identified in literature is that students have a lack of problem solving skills (Apiola & Tedre, 2012) and computational thinking training, both of which are essential to learning programming language concepts which can then be applied to various programming languages (Chetty & Barlow-Jones, 2014). The way that programming is taught in schools affects how students perceive programming and their later ability to understand and use the programming concepts learnt to solve problems. Furthermore,

a programming language also encounters the barrier that second natural languages face in the way that they are taught. Two such obstacles include the focus on writing code instead of reading code and the focus on syntax rather than proper application of the language (Robertson, Lee, & Miller, 1995).

Maltese secondary school students, similarly, to their international peers, enjoy playing digital and video games and challenging themselves to master each and every level (Busuttil, Camilleri, Camilleri, Dingli, & Montebello, 2014). If the motivation that students show when playing games could be transferred to learning programming through game design and play, the potential to learn programming and to understanding the logic needed could be greatly facilitated. Teachers face the challenge of applying knowledge that students are accustomed to, such as playing games, to the more demanding task of writing programs.

This paper attempts to investigate teacher's perceptions on the introduction of programming through game creation by using MIT App Inventor 2. This was the preferred application to base the resource pack upon, since the educational reform in the Computing syllabus was considering using MIT App Inventor 2 as a tool to introduce programming to students in their first year of studying Computing in secondary schools.

The following are the research questions that drive this study:

1. What is the perception of teachers of using App Inventor 2 as a tool to introduce secondary school students to programming?
2. What approaches to the teaching of programming are preferred by teachers?

2. Literature Review

According to Ben-Ari (2016, p. 44), programming "is the formal specification of computation that can be executed by a computer", be it a text-based programming language or a visual programming language. Programming is a key aspect of any Computing course (Ben-Ari, 2016; Preston, 2006), and students following such a course should become competent in this task. Anyone wanting to work in the IT industry needs to "start out as programmers" (Ben-Ari, 2016, p.46). Hence, programming is an essential component of Computing.

2.1. *Difficulties in Learning Programming*

Despite programming being very important to a Computing student, it is the one element of the Computing syllabus that is the hardest to teach and the hardest for students to learn (Apiola & Tedre, 2012; Bennedsen & Caspersen, 2008; Chetty & Barlow-Jones, 2014; Dekhane, Xu, & Tsoi, 2013; Dolgopolovas, Jevsikova, & Dagiene, 2017; Kalelioğlu, 2015; Ma, Ferguson, Roper, & Wood, 2011; Mladenović, Boljat, & Žanko, 2017; Robins *et al.*, 2003; Sáez-López, Román-González, & Vázquez-Cano, 2016; Thomsen, 2008). In fact, at higher levels of education, programming courses have one of the highest drop-out rates globally (Dekhane *et al.*, 2013; Robins *et al.*, 2003).

Research has concluded that a common problem among students is that although familiar with the syntax of a programming language, they are not confident in combining the different structures to form one coherent program (Winslow, 1996). One of the reasons for this is that students are only learning for the sake of passing an exam. Students who learn programming well need "deep level learning skills and problem-solving skills" (Apiola & Tedre, 2012, p. 285; Dekhane *et al.*, 2013). This is not something that can be learnt easily since learning programming is a process (Bennedsen & Caspersen, 2008) and it usually takes a student approximately ten years to become an expert in any given area (Winslow, 1996) – something which cannot be done in a course at tertiary, post-secondary or secondary level alone. Thus, it is axiomatic that to become an expert programmer, one needs a lot of practice and time.

When novice programmers are using text-based programming languages, the focus should be on "language semantics not on syntax" (Mladenović *et al.*, 2017, p. 1485). Hence, ideally, a language with few syntax rules should be used in order to help the learner focus on the important items (Brusilovsky, Calabrese, Hvorecky, Kouchnirenko, & Miller, 1997).

Another reason is that the main skills needed to learn programming, which include critical thinking and problem solving skills were never taught to students in earlier years of schooling (Apiola & Tedre, 2012; Chetty & Barlow-Jones, 2014; de Aquino Leal, A V & Ferreira, 2013; Kalelioğlu, 2015). Thus, it is more difficult for such students to tackle programming tasks effectively.

Students who want to learn programming should ideally have developed "critical thought, problem solving, attention to detail, accuracy and abstract thinking" skills (Chetty & Barlow-Jones, 2014, p. 240). An important skill that encompasses all of these types of thinking is computational thinking – a skill which all students should ideally be equipped with (Dagienė, Pėlikis, & Stupuriene, 2015). Computational thinking, a term coined by Papert (1993), and turned into a household name by Wing years later, is a way to "solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing" (Wing, 2006, p. 33). Computational thinking attempts to take on a problem and deconstructing it into smaller problems which can then be approached more easily.

2.2. *Visual Programming Languages*

Text-based programming languages rely heavily on typing of commands. In the case of Java such commands are case sensitive. Students must not only focus on the logical construction of programs but also keep an eye on the necessary syntax to end statements or to enclose commands within conditional, looping and method structures. All of these syntax rules can hinder learners from focusing on the main aim of programming, that is, problem solving. On the other hand, text-based programming languages can help the learner to write programs very quickly. This can only occur once the syntax and problem-solving processes have been mastered.

Visual Programming Languages (VPLs) can overcome the shortcomings of text-based programming languages. The majority of VPLs, such as Scratch, Alice, and Kodu, use blockly coding (Howland & Good, 2015; Kalelioğlu, 2015). This means that the typing of commands is eliminated and instead all of the available commands are shown on screen and the user simply clicks and drags the command needed onto the instruction screen in order to create a program (Kalelioğlu, 2015), as shown in Fig. 1. This eliminates the possibility of syntax errors, a problem that occurs all the time when using text-based programming languages (Mladenović *et al.*, 2017; Shapiro & Ahrens, 2016). Furthermore, it offers learners several hints such as the types of blocks that fit in certain parameters or commands (Shapiro & Ahrens, 2016). This is very useful for students who have just started studying programming – programming novices – and hence are still new to this area (Howland & Good, 2015; Shapiro & Ahrens, 2016).

While VPLs offer many advantages as well as supporting an active learning pedagogy whereby students are active learners in an environment that stimulates "fun, motivation, enthusiasm, and commitment from the student" (Sáez-López *et al.*, 2016), they also offer some limitations especially for intermediate and advanced learners. Blocks do not give the desired freedom to these learners to edit rapidly, something which can only be done when using a keyboard. Moreover, with more complex programs, the programming commands seem to take up more space on screen when using a VPL (Shapiro & Ahrens, 2016).

Fig. 1. An example of a VPL.

There are many arguments for and against VPLs and text-based programming languages. For the time being, the IT industry is still using text-based programming languages for software development. Some researchers have suggested that more research should be done on the development of translation software that translates the code of a program created by a VPL to a text-based programming language for easier transition (Shapiro & Ahrens, 2016). Others have suggested that students should start off by learning programming by using a VPL and then transitioning to a text-based programming language at a later stage (Computing at School, 2013; Shapiro & Ahrens, 2016).

For the time being, at some time or other aspiring prospective programmers will need to learn how to program using text-based programming languages (Shapiro & Ahrens, 2016). Students learning programming by using a VPL should be made aware that they cannot rely forever on the use of blocks (Shapiro & Ahrens, 2016) since most companies in the IT industry do not use VPLs.

2.2.1. *MIT App Inventor 2*

MIT App Inventor 2 (AI2) is a free "visual, drag-and-drop tool for building mobile apps on the Android platform" (Wolber, Anderson, Spertus, & Looney, 2014, p. xiii) which can then be uploaded to the Google Play Store (Dolgopolovas *et al.*, 2017). It offers the user several features to design a user interface according to what the user desires to create. The items that the user has included in the user interface are controlled by blocks – which is the equivalent of text-based programming in other programming editors, as shown in Fig. 2.

2.3. *Programming Pedagogy*

Programming pedagogy refers to how programming is taught to students at various educational levels. This theme has fascinated researchers and teachers alike since programming is the hardest component to teach in a Computing course (Apiola & Tedre, 2012; Bennedsen & Caspersen, 2008; Chetty & Barlow-Jones, 2014; Ma *et al.*, 2011; Robins *et al.*, 2003; Thomsen, 2008). If the methods chosen to teach programming are varied to meet different learners and to help motivate the students, the students could be further encouraged to develop their programming skills.

The traditional approach for teaching programming, has always been of an objectivist nature whereby the students sit, listen, and try to absorb whatever the teacher was

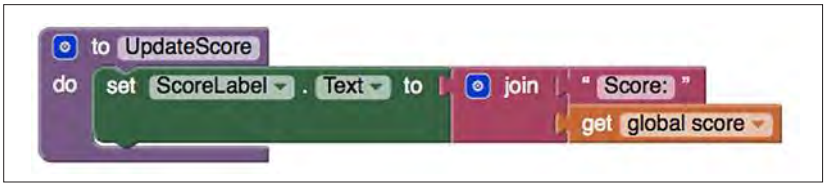

Fig. 2. Programming Blocks in MIT App Inventor 2.

saying (Van Gorp & Grissom, 2001). Even in less hands-on subjects, this can be deemed as boring since the student is a passive learner. In a very practical subject such as programming, it is useless to listen only. One must regularly 'get their hands dirty' in order to learn and progress.

Constructivist learning theories suggest that in order for students to learn programming, they need to be presented with activities in which they are active participants and in which their kinaesthetic senses can be engaged (Sentance & Csizmadia, 2015). In order for the students to become active participants, they first need to be provided with context – a task that offers a problem in a simplified way – construction – whereby the student is able to "construct knowledge based upon meaningful activities" – and collaboration – here the student works with others to examine and understand different views about the same problem and so be able to sharpen and enhance their own ideas (Van Gorp & Grissom, 2001, p. 248). In this way, learning has shifted from teacher centred to student centred (Van Gorp & Grissom, 2001).

The following are some constructivist teaching strategies used in programming education as outlined in literature:

- Developing an algorithm to solve a problem (Sentance & Csizmadia, 2015; Van Gorp & Grissom, 2001) which allows students to understand a problem and provide a solution for it. This increases familiarity with the flow of control of a programming problem.
- Code walkthroughs (Sentance & Csizmadia, 2015; Van Gorp & Grissom, 2001) which allow students to start becoming familiar with a given programming language and gives students the opportunity to read, analyse and interpret readymade programs.
- Code debugging (Van Gorp & Grissom, 2001) which allows students to look more in depth at given code by spotting and resolving syntax, logical and runtime errors.
- Lecture note reconstruction (Van Gorp & Grissom, 2001) which focuses on students on becoming actively involved in creating their own notes by engaging their listening skills and short-term memory.
- Metacognition (Chetty & Barlow-Jones, 2014; Sprankle & Hubbard, 2012) which requires students to use advanced thinking skills to evaluate their solutions to analyse whether they are effective, efficient and workable.
- Pair Programming learning (Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011; Lau & Yuen, 2009; Preston, 2006) which is a type of collaborative learning and can be used in conjunction with the above-mentioned strategies. It allows students to work in pairs by alternately taking on the role of a driver (actually writing the code) and observer (oversees driver, looks out for mistakes and suggests improvements to the code).
- Active learning pedagogy (Brown, 2006) is used with any of these teaching strategies. Students are presented with all the necessary material, and then they decide themselves when to learn.

In this study, these constructivist teaching strategies in conjunction with game creation were proposed to introduce programming to novice programmers.

2.4. *Learning Programming through Creating Games*

Salen & Zimmerman (2003, p.80) define a game as "a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome." Digital games are similar to traditional ones but within the context of real and virtual lives. In other aspects, digital games are similar to conventional games in that they also involve "making choices and taking actions" (Salen & Zimmerman, 2003, p.33). According to Salen & Zimmerman (2003), such a game is categorised by four main elements: objects, attributes, internal relationships, and environment.

Fig. 3 explains how these four elements are essential to a game being a game according to the definition of Salem & Zimmerman (2003).

Games have also been proposed to teach programming in various studies (Busuttil, 2014; de Aquino Leal, A V & Ferreira, 2013; Kumar & Khurana, 2012) since it has been shown that games do increase student's motivation and understanding of the concepts being proposed in class. Games are very visual tools that can aid students to visualise certain programming concepts "through the use of graphics and animation" (Ma *et al.*, 2011, p. 63). They also help students to visualise more abstract ideas in programming and to create a "mental model" of such concepts (Margulieux, Guzdial, & Catrambone, 2012). The drawback with many of these visual tools is that they have not been used from a constructivist point of view and thus treat the student as a passive learner (Ma *et al.*, 2011).

Some studies seem to propose a gamified approach to learning programming, as discussed formerly, whereby the topics to be learnt are presented in the form of a game

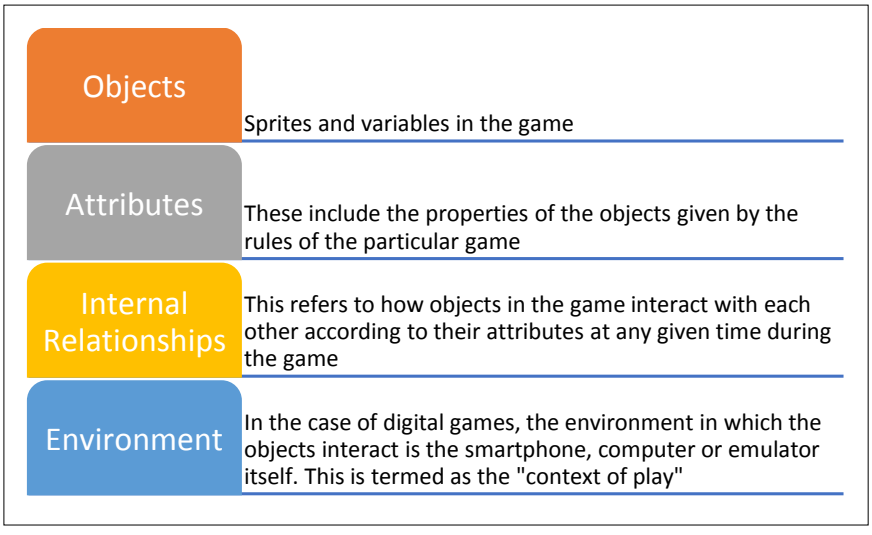

Fig. 3. The Elements of a Game.

with different levels, tasks, and extra quests. Points and badges are earned through the game and when the game is completed, the student would have gone through the learning process by using a non-traditional approach (Kumar & Khurana, 2012).

A study conducted by de Aquino Leal & Ferreira (2013) offers a refreshing view of how one can use actual sports games to teach programming concepts. In their study, the game of football, which all the students knew how to play is considered by the class and programming patterns related to it are extracted. These can include loops, counters, determining the team with the highest points, and so on. This helps students to relate programming concepts with something that they are already familiar with.

Today, games are not only played on desktop computers but even more so on portable devices such as tablets and smart phones. Such devices are quite powerful and have become cheaper in recent years (Dekhane *et al.*, 2013; Hsu & Ching, 2013).

Many frameworks have been developed in this regard to help students to learn programming such as Scratch, Alice, Lego Mindstorms, and Kodu (Dekhane *et al.*, 2013; Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015). These frameworks rely on blocks which the users drag and combine together in order to write a program. Hence the element of worrying about programming syntax is eliminated (Ouahbi *et al.*, 2015).

Websites such as code.org were created with the aim to get people to learn how to code by solving puzzles through relatively easy programming. Most of these puzzles made use of other interfaces such as Scratch, which allows users to drag and drop commands. In other games, users have to type in commands to solve a level. The available commands are however shown on the side in order to help out the user. Such websites are great motivators to help students to learn programming concepts since they make use of highly visual tools which include graphics and animations as opposed to the traditional text only approach to teaching and learning programming. Such a platform was found to help students improve their self-confidence and "problem-solving ability" as well as their "mathematical and geometrical knowledge" (Kalelioğlu, 2015, p. 207).

Some complex games have indeed been developed to help teach programming such as Code Warriors which can be played in single-user mode or multi-user mode and presents the player with different puzzles whereby robots engage in battle by writing JavaScript commands. This game helps players to develop logical thinking and to learn JavaScript by progressing through different levels that range from Beginner mode to Warrior mode.

Other software specifically designed to create mobile applications with relatively easy programming has also been developed. The MIT App Inventor 2 (MIT AI2) is one such software that allows the user to create mobile-based applications that are compatible with Android phones. If one is not in possession of a phone equipped with this operating system, an online emulator exists for immediate testing. The MIT AI2 is

a free software that uses a drag and drop feature to select commands that will control items on the screen (Hsu & Ching, 2013; Margulieux *et al.*, 2012). Thus, the user does not need to remember all the commands by heart or make mistakes in the programming languages' syntax. In this way, students are able to learn the programming concepts without learning a particular programming language. They can then extract and apply this knowledge to any programming language that they learn later on.

Such progress can have positive repercussions in education. In one way, teachers of any subject can easily create an educational mobile application for their subject to help their students to learn more (Hsu & Ching, 2013). Teachers should exploit this interest in such devices and take advantage of them to teach their students (Dekhane *et al.*, 2013; Karakus, Uludag, Guler, Turner, & Ugur, 2012).

From another perspective, programming students who are creating mobile applications, have intrinsic motivation since they are learning to program a contemporary item. This makes learning very much related to their everyday lives and much more interesting. Thus, learning becomes fun.

On the other hand, prospective programmers may be misled and think that they do not need to understand and learn a proper programming language in order to create a successful application (Dekhane *et al.*, 2013).

Students can learn to create applications by being assigned programming tasks, peer review and by keeping a reflection journal (Hsu & Ching, 2013). The constructivist teaching methods discussed in the previous chapter work hand in hand with such an approach. In this case, the MIT AI2 software merely serves as the framework in which programming concepts are learnt by presenting students with various activities inspired by constructivist approaches.

Through the teaching of writing mobile applications students can benefit from "immediate visual feedback" and can help to improve their problem-solving skills (Dekhane *et al.*, 2013, p. 307). Not much research has been carried out to determine how the development of mobile game applications can help students to learn programming concepts. This study attempts to fill that void.

2.5. *The Resource Pack*

A resource pack was created for this study which included several activities aimed to introduce programming to computing novices by creating mobile-based games using MIT App Inventor 2. These activities were evaluated by a group of Computing teachers.

To better understand what this resource pack consists of and how it aims to deliver on teaching programming basics by using the constructivist teaching strategies, the Table 1 has been created to summarise its contents. This will help to understand better what the Computing teachers were evaluating. The tasks in the workshops were purposely scaffolded.

Table 1
Resource Pack Activities

Task Name	Constructivist Method/s Used	Description	Games Used and/or Developed
Deriving Rules	Planning a Program	Students are asked about games and what types of games they play. The rules of any game students are familiar with are derived.	N/A
Understanding a program	Code walkthroughs Pair Programming	Students are presented with a readymade game found on the AI2 gallery: Ping Pong. Students explore the code and understand what parts of the code is doing.	Ping Pong (Fig. 4)
Understanding and solving part of a problem	Develop code from algorithms Insert comments into existing code Pair programming	Students are presented with an incomplete game: Mole Mash. Students add the appropriate commands in order for the game to function properly.	Mole Mash (Fig. 5)
Finding Errors	Code debugging Code walkthroughs Pair programming	Students are presented with a Minigolf game which is purposely riddled with errors. Students work in groups to locate and fix the errors in this program.	Minigolf (Fig. 6)
Creating a Solution	Planning a Program Develop code from algorithms Code debugging Pair Programming	Students are presented with the rules of the 'Ladybug Chase' game including a readymade design framework for this app. Students work in pairs to develop the appropriate code to construct a working program, based on the given rules.	Ladybug Chase (Fig. 7)
Designing a Game	Planning a Program Code debugging Develop code from algorithm Metacognition Pair Programming Active Learning	Students are presented with the game description of a new game: Brick Break. Students are asked to extract the rules of the game, design the game and program it.	Brick Break (Fig. 8)
Creating a Game	Planning a Program Code debugging Develop code from algorithm Metacognition Pair Programming Active Learning	Students decide on a game they want to program. They should provide the game description and rules and then design and program the game.	Students develop any game that they like

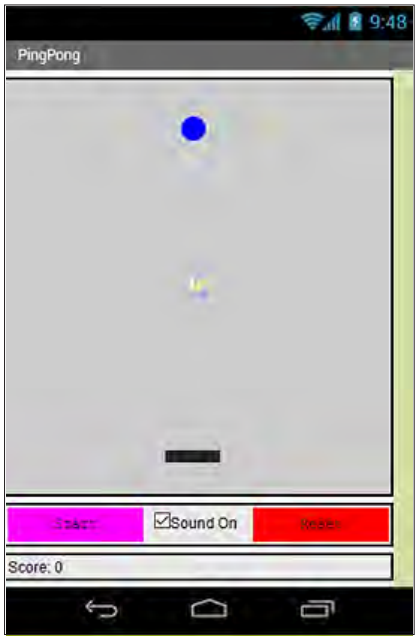

Fig. 4. Ping Pong.

Fig. 5. Mole Mash.

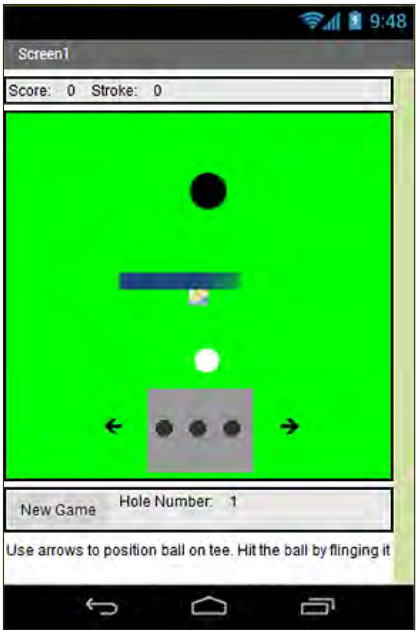

Fig. 6. Minigolf Game.

Fig. 7. Ladybug Chase.

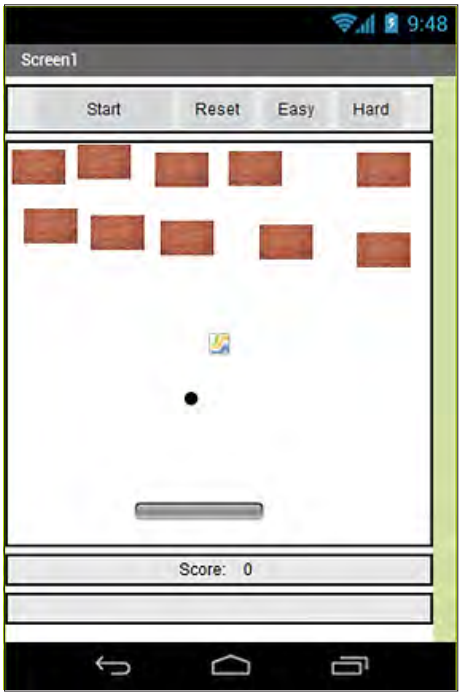

Fig. 8. Brick Break.

3. Methodology

Qualitative research seeks to get an in-depth view of a person's perceptions rather than a wider interpretation (Haralambos & Holborn, 2004). The qualitative research method was adopted as the main research method in thus study, in particular the use of case studies. A case study is an ideal approach to gather "in-depth, multi-faceted understanding of a complex issue in its real-life context" (Crowe *et al.*, 2011, p. 1). Also, a case study is helpful when observing as a third party and seeing a somewhat unbiased view of the situation, which will allow one to understand better what can be improved in the given situation (Crowe *et al.*, 2011).

Case studies were an ideal method of data collection in this research since in-depth data about teachers' perceptions of teaching programming through game creation and evaluation of the resource pack was needed.

3.1. *The Participants*

Computing teachers in secondary schools were asked to participate in a day workshop at the end of the scholastic year, to evaluate the resource pack and to obtain their

Fig. 9: Teacher Workshop Information

perceptions on the teaching of programming through mobile game creation. These teachers were sought out by using an online contact form which was distributed to all the heads of schools in Malta, after obtaining the necessary permissions from the Education Department, the Secretariat for Catholic Education and the respective heads of private schools. The contact form was also distributed on the Computing and IT Teachers Malta Facebook page, which gave information about the workshop, as shown in Fig. 9.

Out of a total of approximately 85 Computing teachers in secondary schools, six teachers agreed to participate in this case study over the course of a day workshop held at the University of Malta. Another computing teacher carried out this workshop, to allow for observation.

Four of the six participants were male and the other two were female. Two teachers worked in state schools, three others in church schools while the other teacher worked in a private school. The teachers had a variety of teaching experience in Computing. One teacher had been teaching for over twenty years, two others had been teaching between fifteen and twenty years, two others for between ten and fifteen years and another for less than ten years.

3.2. *The Workshop*

During the workshop, the resource pack was reviewed one activity at a time, and participants had the ability to create mobile-based games using MIT App Inventor 2. During these activities, discussions regarding the current practices of teaching Computing as well as the impending education reform took place. All of the discussions were relevant to the teaching of Computing at secondary level.

3.3. *Data Collection*

Data from the workshop was collected by digitally audio recording the entire workshop. This allowed for better data analysis. The researcher also kept a research journal with observations.

3.4. *Data Analysis*

The audio recording of the workshop had to first be transcribed. This was done soon after the workshop so as to correctly identify persons who were speaking. Once the transcripts were ready, they were analysed by means of a Computer Assisted Qualitative Data Analysis Software (CAQDAS), in this case NVivo.

Each transcription was re-read and coded by the themes that were prevalent. Once all of the transcriptions were coded, a document for each identified theme was extracted from this software. The themes with the highest frequency were deemed to be the most important. Themes which were the least recurring were re-categorised under a similar theme which had a higher frequency.

The researcher's journal observations were also inputted in the CAQDAS software and coded as well. The themes which emerged where compared with those emerging from the teacher workshop.

4. Findings and Discussion

The teacher workshop resulted in several recurring themes which were condensed into seven main areas, in order of frequency. These are discussed and compared to literature in this section.

4.1. *Blocks-based vs. Text-based Programming Languages*

An issue that dominated the teacher workshops was the choice of programming language that one should introduce programming to students with, in particular whether a blocks-based programming language or a text-based programming language would be

more appropriate. The discussion did not reach a consensus due to the different perspectives that the teachers had on the matter.

Nonetheless, all the teachers agreed that using an interface such as MIT AI2 would attract students immediately due to its visual nature as opposed to a text-based language such as Java. This also allows for immediate experimentation and testing, since there is no need to learn commands beforehand in order to create something, unlike a text-based programming language. Other teachers also noted that using a graphical user interface is more appealing to the students this advantage of using a blocks-based programming language.

Other teachers, while seeing all of these advantages, reflected on whether secondary school was the best place to introduce blocks-based programming languages. Teacher 2, in particular, questioned this notion and suggested that such a programming language was more appropriate for primary school children, who would benefit largely from learning the basic programming constructs from a young age.

Another strong argument against text-based programming languages was made by Teachers 3 and 6, who were perplexed about the reality that in today's age everything has become visual and thus uses of graphics. Today's students are millennials and they are accustomed to learning visually. Teacher 3 in fact questioned why when teaching programming using textual languages, most of the exercises seem to be related to creating text-based interfaces. This contradicts the multi-modal nature of the world around us. This might make it harder for students to relate to programming. No graphical elements are used when teaching Java. The points raised by the teachers about the nature of the programming language chosen to teach students programming is similar to what has been found in literature. Research has shown that VPLs offer more motivation and enthusiasm for students as opposed to a text-based programming language (Sáez-López *et al.*, 2016) while focusing on the actual problem solving rather than the syntax of the language, which is highly beneficial to programming novices (Howland & Good, 2015; Kalelioğlu, 2015; Mladenović *et al.*, 2017; Shapiro & Ahrens, 2016).

The teachers' concerns about whether blocks-based programming languages were appropriate for secondary school children where similar to other researchers' concerns that blocks-based programming languages may be too limited for intermediate and advanced learners (Shapiro & Ahrens, 2016). Hence, it could be suggested that some secondary school students quickly outgrow blocks-based programming languages. The suggestion that a fusion of blocks-based programming language and a text-based programming language is used was also put forward and has also been suggested in literature to exploit the advantage of both (Shapiro & Ahrens, 2016). Another concern that also emerged in literature was that students may think that using a VPL is enough to create an application and would thus be misled into thinking that they do not need to learn a proper programming language (Dekhane *et al.*, 2013).

From this fervent discussion regarding the choice of programming languages, it was obvious that teachers were in favour of students learning programming concepts by using a visual programming language, without excluding the possibility that students might be exposed to a text-based programming language once those concepts have been mastered.

4.2. *Catering for Different Student Abilities*

Teachers noted that the MIT AI2 interface benefits high achievers more than average or low achievers. Teacher 1 voiced concern that if students see the code from the very first lesson, they would "define programming as something difficult and unattainable." Teacher 3 in fact suggested that students are not shown any code at first but are instead exposed to examples of the games that can be created by using MIT AI2 while keeping the students in a reality check that to be able to create such games, there are many steps that needed to be taken along the way. Students also need to be told that to create a game, different aspects of the game need to be tackled one by one, and not all at one go.

Teacher 6 mentioned that students had different abilities, with some students being able to carry on with minimal direction while others were not able to under any circumstance despite trying to introduce concepts through games, group work etc. Teachers were also concerned about high achieving students who found the programming aspect easy. Although only a few students fall in this category, teachers were concerned that such students could become easily bored and become demotivated. A Computing school, similar to the sports school, was suggested to cater for students who were good at Computing-related subjects and who could flourish abundantly in a specialised environment.

Teacher 5 voiced concern over the issue that some of the students have serious literacy problems and as such, an interface like MIT AI2 would be too difficult for them since it would not be at their appropriate level. In fact, these students often use Scratch to program, which offers simpler words, and usually do not go past a VPL. This teacher spoke of these students' probable attitudes about using such a tool. Most probably, these students "would rather do nothing, rather than fail." This teacher also ruled out the possibility of introducing something basic for all and then giving extra tasks to those who succeeded. This teacher noted that from past experience such an approach would not work since the students would not perform well on purpose so that they would not be given any extra or more challenging tasks. Furthermore, this teacher also pointed out the importance of trying to reach the students at their own level and to empathise with them:

"You need to show them that you understand that the concept you're explaining is not easy, otherwise they will simply shut down."

The concerns that this group of teachers had regarding the suitability of an IDE such as MIT AI2 for all students has also been voiced by other researchers who acknowledge that high ability students perform better in programming tasks, while lower ability students need further support to attempt to achieve the same task (Lau & Yuen, 2009). Other issues that were raised about this theme are mostly pertinent to the situation in Maltese schools.

4.3. *Catering for Different Student Scenarios and the Class Setting*

A common theme which also emerged in other research, is that today's generation of students are living in highly different times and situations from previous generations, and as such, require different support and approaches to education (Prensky, 2001). This is not to mention the fact that each individual learns in different ways, and thus, different approaches to teaching need to be adopted to try and target as many learners as possible (Mladenović *et al.*, 2017).

Generation differences also play a factor in the ever-changing student scenarios. Some teachers who have been teaching for a longer number of years could easily compare the situation of a few years ago with that of today. The situations of our students are far from linear in terms of family life, activities undertaken after school and time management, to mention only a few. These differences are challenging for teachers to plan homework, which is not being given importance by all the students. A suggestion of a flipped classroom concept by giving students video tutorials to watch was made by the researchers, who experienced increased student motivation from the first part of this study, which is not discussed here. Teacher 2 however pointed out that this was not possible with students who do not do anything at home and as such, similar activities need to be included during lessons. Hence, the teachers cannot always rely on students doing their work at home, and then building upon that work during the next lesson. Teacher 2 remarked that with the different family aspects in today's society, students do not always sleep in the same house, and as such may have their resources, including books and computer, in different locations, making it impossible for them to work. Hence, the work that is done at home "should be over and above" what is done in class.

Another factor that hindered the proper delivery of programming concepts was class size. Teacher 5 noted that the number of students in class did not allow for individual attention especially when introducing more abstract concepts.

Possible solutions to help low ability students emerged. Teacher 5 suggested that students who have difficulty in reading and writing or who are low ability students, would benefit more if they used Scratch.

A different perspective was offered by Teacher 1 who acknowledged that at secondary level, some students are still developing and may be somewhat immature to take on studying or programming seriously. Hence age could also be a barrier for students.

4.4. *Problems with App Inventor 2*

Teachers were not familiar with the syntax of the interface and although they knew what had to be done, they had a hard time finding the appropriate commands. The fact that each object had its own set of commands had to be explained. Furthermore, the use of the canvas also had to be clarified to the teachers. Teacher 1 remarked that "you have to learn the interface more than learn the language." Other teachers also remarked that students would ultimately find some concepts to be abstract as well even though graphics

were being used. The reasoning behind controlling certain events in the game required abstract thought.

One of the most difficult concepts that the teachers themselves found challenging in their initial encounter with MIT AI2 was the clock timer which allowed for simultaneous events by checking statuses and updating variables or different elements of the game at the same time. This issue also related to the mathematical aspect that is needed to program various events and which the teachers felt that the students may find difficult to accomplish. On a similar note, Teacher 6 noted that writing an equation using Java was done much quicker than by using a drag and drop interface. Hence, time could be wasted by dragging the appropriate blocks instead of focusing on the equation that needs to be written.

Some teachers were concerned that novice programming students who were introduced to such an interface would feel lost since they would not have awareness of certain things. Teacher 6, in fact, mentioned an example whereby the teacher would be explaining what an increment was, and a student would still be trying to figure out from where a variable can be created in the interface. Teacher 6 also commented that students are shrewd enough to calculate that if a simple game required a good amount of code, then more complex games required more programming. This could ultimately result in students 'fearing' code due to its never-ending commands.

Another challenge posed by MIT AI2 was the way that errors are shown to the user. Each time an error occurred, a window would appear notifying you and while in the process of arranging said error, that same window would reappear every few seconds until no more errors are detected. Such an approach was frustrating for the teachers and they in turn could see that this would also be frustrating for the students alike.

A main problem of MIT AI2 was its setting up especially the use of the emulator. The initial part of setting up was found to be easy by the teachers. The problems ensued with the advent of the emulator which was not working properly for all the teachers despite a good Internet connection. Concern was raised that since some emulators took a while to run properly, should a similar scenario occur in class, some students would get bored and might cause unnecessary disruptions in class. Teacher 6, who had previously used MIT AI2, commented that programs created using this interface were better tested by using a mobile phone since testing was much faster. This was however not an ideal solution in schools, were mobile phones are not allowed in class. Furthermore, as already stated previously, mobile testing can only occur on an Android phone which is on the same Internet connection as the computer being used.

Teacher 2 noted that while MIT AI2 offered the 'wow' factor of creating mobile apps, the same could be achieved academically by using Scratch. In Teacher 2's opinion, Scratch was more user friendly and less time consuming to set up when compared with MIT AI2. Teacher 1 concurred with this and went on to point out that using MIT AI2 in class would require a good infrastructure, ideally a mobile to test programs within class in lieu of the emulator as well as a thorough training course for teachers. Teacher 6 also agreed that students would be thoroughly engaged with using such an interface. However, concerns were also raised whether using such a tool would be enough for students who wanted to work in the industry. Even though novice programmers are still at the

start of their journey in programming and will have training over a number of years, a smooth transition between blocks based and text-based IDEs is much needed.

Other researchers had also found that some features of a block based language like MIT AI2, although having a lot of benefits, also had their limitations, such as the time-wasting needed to find the commands and assemble them as opposed to type in the needed commands (Shapiro & Ahrens, 2016).

Teachers saw the potential of MIT AI2 but since its implementation is not as smooth as other interfaces that the teachers previously used, proper preparation for use in class is needed. Some problems are easy to deal with, while others are more difficult since they are hard-wired in the interface. Yet again, other require the teacher to find a way to deliver abstract concepts to students. This latter problem is present in all programming editors and is always a challenge.

4.5. *A Stronger Framework for Programming Education*

Teachers 1 and 2 seemed put forward the idea that programming should also be taught at primary school level. A blocks-based interface is ideal to be introduced at primary level since children are still learning how to read and write. However, this does not prevent them from being able to use their thinking faculties to solve small problems through programming.

The teachers suggested that Scratch can be introduced at primary school level, which then transitions to MIT AI2, which could then further transition to a more advanced editor such as Visual Studio. The ideology of these teachers was that if students are well versed in programming concepts by the time they finish primary, then they can easily take on more challenging tasks in programming, since they would no longer be novices to it, but rather they would be enhancing their knowledge and applying it further.

Teacher 2 went on to note that in this regard the education system may have failed since students were not being well prepared for programming and instead of advancing in the subject, we seem to have lowered the level. Furthermore, if things "are being done in the way they are supposed to, the students would arrive already knowing how to do basic programming at Year 9 and then we move on with Java." Teacher 3 agreed with this and remarked that abroad, in schools where programming is introduced from a young age, the students flourished in other subjects as well, since they were actually being challenged to think and to solve problems.

The shared proposal between the teachers attending the workshop and other researchers is that students should be exposed to learning situations where they can develop their problem solving skills from a young age (Apiola & Tedre, 2012; Chetty & Barlow-Jones, 2014; de Aquino Leal, A V & Ferreira, 2013; Kalelioğlu, 2015). These skills reach their culmination in the acquisition of computational thinking skills which are vital for any person wishing to learn programming (Computing at School, 2013; Sentance & Csizmadia, 2015; Wing, 2006). The younger a person starts to acquire these skills, the more prepared that person will be to tackle programming problems and attempt to solve it by considering various approaches.

It was evident from the teacher workshops that more needs to be done in terms of preparing students to take on programming. This issue needs to be dealt with from primary schools, especially since the skills acquired through computational thinking, such as problem solving, can be applied to all areas of our society and not just programming.

4.6. *Scaffolding*

Another theme that emerged in the workshop was that of scaffolding in order to support students in their learning of new concepts and ideas.

An idea that seemed to resonate with most teachers was that an interface such as MIT AI2 could be introduced to students in a project like manner.

Teacher 6: We can move in baby steps. It's better to tackle one piece at a time…for example, first I code the ball to move on its own – I simply focus on it. Then I start another piece of the program, then another, in order to reach the required level.

Teacher 5: As an introduction to using Scratch, I usually ask students to think of a game such as Ping Pong. Then I give them small tasks like changing the background of the game to a football net, adding a goalkeeper of their choice and editing it using Scratch itself, and then adding a ball. For them that is already an accomplishment.

In fact, some teachers start introducing this concept immediately to their Year 9 students from their first programming lesson. They ask students to think of the games that they play, such as FIFA. However, it is explained that at this level they cannot recreate something similar since there are many things that they need to learn first. Mr Brown also suggested that students should be made aware that the programs and apps they use were actually coded by people who did not program that app on their first attempt. Instead they had started with something more basic and gradually advanced to more complex programs. Teacher 2 continued with this suggestion and recounted how in introductory programming lessons, students would be able to recreate simple methods for games that they play. An example was that of Prince of Persia, whereby students could write a method for when the Prince is hit, so that the life value is decremented. The element of the game factor could really encourage the use of scaffolding in programming lessons. This method for introducing programming to students has also been found in literature whereby students tend to be more enthusiastic and keen to learn programming when it is presented by means of activities in which they already actively participate in (Sentance & Csizmadia, 2015). Hence, they can relate to programming much better.

Scaffolding was found to also happen by teachers giving video tutorials to students so that they are supported at home too as well as giving students reference sheets with the basics of a programming language. Teacher 2 proposed the idea of scaffolding in a different manner by using Scratch as a tool for scaffolding. Students would at first use Scratch to control sprites by doing simple things. Once they have enough knowledge of how to do basic coding using Scratch, they are able to move to a similar but more complex interface, that is, MIT AI2. Teacher 3 went on to further this concept by suggesting that MIT AI2 could also serve as a scaffolding for text-based editors.

The use of scaffolding amongst programs created with MIT AI2 itself was also mentioned. The teachers noted that there were elements common to many games programmed with MIT AI2. Thus, these elements could be used in new programs and could help to serve as a guide to the students.

The use of scaffolding to teach programming has been suggested in literature (Margulieux *et al.*, 2012; Van Gorp & Grissom, 2001) whereby students are supported with the necessary material in order to reach some higher goal. As is evident from the discussion of this theme and the research explored in literature, the concept of scaffolding is used in conjunction with other teaching methods such as pair programming and code walkthroughs.

4.7. *Teacher Training and Peer Discussion*

The teachers were concerned that if MIT AI2 or something similar were to be implemented in the syllabus, a lot of training would ensue especially since the nature of such a programming language was quite different from the sequential and object-oriented nature of Java. Teachers also noted that even if the training is well planned and implemented, there would still be some teachers who would still resist such ideas.

The teachers also agreed that there is much needed discussion between Computing teachers in Malta. It was evident from the workshops that most of the teachers were thinking on the same lines but were rarely getting an opportunity to meet with their colleagues and discuss pertinent issues. Computing teachers only meet formally once a year during in-service course – when these are held – and in which new reforms are usually introduced and discussed and then are never implemented from a higher level. Hence, the teachers felt that they need to be at the forefront of discussions since they were the ones dealing with students directly.

5. Evaluation and Recommendations

During the teacher workshops, teachers agreed that different methods need to be used with different learners. The focus however was more on the tools used to teach programming rather than the specific teaching methods outlined in literature. Nevertheless, the tools that teachers use, in themselves aid students to learn. The way that such tools are presented to students will affect whether students learn a particular tool or learn general concepts which can then be applied to similar tools as well. The latter scenario would have employed better use of teaching methods and will have equipped the students with better adapting strategies than the former scenario. It is up to teachers to understand which teaching methods are best suited for their students and to continue informing themselves about the different approaches that other teachers, both locally and abroad, employ to deliver effective programming concepts to their students.

The final research question of this study was to determine whether the teaching process for teachers was facilitated when programming was introduced through games. The teacher workshops carried out during this research aimed to answer this question. The teachers saw an advantage to using games to introduce programming and anticipated that students would enjoy programming lessons done in this way. They predicted that their students would be greatly motivated to learn programming in such an exciting manner. The teaching process in terms of preparation would however increase since teachers would need to be trained to teach in this manner as well as to prepare various resources which are appropriate for such a manner of teaching programming. Furthermore, many teachers would need to change their manner of delivering concepts to students and instead provide guidance to their students in such a way as to help them construct the knowledge that would traditionally have be spoon-fed.

Although the teachers saw that a task for programming a game would be better received by students than writing a program that calculates numbers or emulates a quiz, they were unsure regarding the interface used. They anticipated that high achieving students would find little problems in using the interface. However, they were concerned that students of lower ability would struggle to cope. They suggested that other blocks-based interfaces are explored, which ideally made less use of mathematics and that are simpler to use so that these students could also be introduced to programming at the appropriate level. A strong point that emerged from this research was that a radical change in the educational system needed to occur to provide a solid foundation for the education of programming. All the teachers felt quite strongly that students were not equipped with computational thinking skills or problem-solving skills, and this was hindering their capacity to solve simple programming tasks. This tied closely with the fact that students were not being exposed to programming at primary level, something that the teachers attending the workshop felt that students would benefit a lot from, even if they did not choose to pursue Computing.

Thus, from this research, it has become apparent that teachers yearn to have new and innovative ways to introduce programming to students, which would stimulate their learning. The current situation is far from ideal. While some teachers continually reinvent their lessons to meet their students' needs, others seem to be stuck in more traditional approaches to teaching programming. A common national approach needs to be taken if we are to raise a generation that is capable to write simple programs in the same way that they are capable of playing and winning a video game.

Through the course of this research, we reflected about various possibilities as to how programming education in Malta could be enhanced. The following are some suggestions, which may require further research to properly evaluate and determine whether they would be effective:

- Introducing computational thinking skills and problem-solving skills to primary school students.

- Programming should be made available for all Guzdial (2015) to study, throughout compulsory schooling, regardless of the career path the student will choose later on.
- Further training of Computing teachers in programming.
- Sharing of good practices by Computing teachers.
- Catering for different student ages and abilities by using different IDEs.
- Creating a relevant programming education programme which is regularly updated to meet today's needs and learners.

7. Conclusion

Summarising, the highlights of this research are as follows:

- Teachers agree that using blocks-based programming languages, such as MIT App Inventor 2, to introduce programming to students increases student motivation rather than using text-based programming languages.
- Teachers agreed that blocks-based programming languages may be better suited to younger students.
- While MIT App Inventor 2 is a great tool to teach programming concepts to students, it has its limitations.
- Programming education needs to have strong foundations based on logical and computational thinking from a young age.
- The use of constructivist approaches to teaching programming is highly important and useful.
- Teachers agree that more teacher training with regards to teaching programming should take place regularly.

Programming has always been at the heart of Computing and in order for it to attract more students to pursue it, the way that it is taught needs to remain relevant to our students. Education needs to evolve to meet the current situations and it is the duty of each and every educator to provide the best possible learning experiences for their students. In the area of programming education, we need to do more. More research is needed to understand different and effective ways of teaching programming. Furthermore, programming lessons should not be restricted to learning specific languages but should allow students to be presented with various problems which they can solve together as a team and which they may program eventually. Programming should be seen as a tool to create solutions for the world and as something exciting, not something boring which results in a few lines of text as output. We need to keep programming education relevant to our millennial students. We would be doing them a disservice if we do not integrate normal day-to-day technology in programming lessons.

References

Apiola, M., & Tedre, M. (2012). New perspectives on the pedagogy of programming in a developing country context. *Computer Science Education*, 22(3), 285–313.

Ben-Ari, M. (2016). In defence of programming. Paper presented at the 20th *Conference on Innovation and Technology in Computer Science Education*, Vilnius, Lithuania. , 7(1) 44–46.

Bennedsen, J., & Caspersen, M. E. (2008). Exposing the programming process. In: J. Bennedsen, J., Caspersen, M. E., & Kölling, M. (Eds.). (2008). *Reflections on the Teaching of Programming (1st ed.)*. Berlin Heidelberg: Springer-Verlag.

Brown, C. (2006). An active learning pedagogy in a programming course. *Issues in Information Systems*, 7(1), 124–128.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P. (1997). Mini-languages: A way to learn programming principles. *Education and Information Technologies*, 2(1), 65–83. DOI: 1018636507883

Busuttil, L., Camilleri, L., Camilleri, V., Dingli, A., & Montebello, M. (2014). Digital and video game usage in Malta. (). *University of Malta: gamEd* (Games in Education). Retrieved from `http://www.digital-gamesmalta.com/digital-games-report`

Busuttil, L. (2014). I want to be a game maker: Experiences of digital game making with eleven year olds Retrieved from `http://etheses.whiterose.ac.uk/7593/`

Chetty, J., & Barlow-Jones, G. (2014). Novice students and computer programming. *Mediterranean Journal of Social Sciences*, 5(14), 240–251.

Computing at School. (2013). *Computing in the National Curriculum – A Guide for Primary Teachers*. (). Retrieved from `http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf`

Crowe, S., Cresswell, K., Robertson, A., Huby, G., Avery, A., & Sheikh, A. (2011). The case study approach. *BMC Medical Research Methodology*, 11, 100.

Dagienė, V., Pėlikis, E., & Stupuriene, G. (2015). Introducing computational thinking through a contest on informatics: Problem-solving and gender issues. *Informacijos Mokslai*, (73), 55–63. Retrieved from `http://www.ceeol.com/search/article-detail?id=467909`

de Aquino Leal, A V, & Ferreira, D. J. (2013). Teaching computer programming based on patterns with activities and collaborative games using concrete materials for high school students. Paper presented at the *Frontiers in Education Conference, 2013 IEEE*, 1604–1610.

Dekhane, S., Xu, X., & Tsoi, M. Y. (2013). Mobile app development to increase student engagement and problem solving skills. *Journal of Information Systems Education*, 24(4), 299–308.

Dolgopolovas, V., Jevsikova, T., & Dagiene, V. (2017). From android games to coding in C – An approach to motivate novice engineering students to learn programming: A case study. *Computer Applications in Engineering Education*, 26(1), 75–90. DOI:10.1002/cae.21862

Guzdial, M. (2015). *Learner-Centered Design of Computing Education*. San Rafael: Morgan & Claypool Publishers. Retrieved from `http://ebookcentral.proquest.com/lib/[SITE_ID]/detail.action?docID=4205923`

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education*, 21(2), 135–173.

Haralambos, M., & Holborn, M. (2004). *Methodology. Sociology Themes and Perspectives* (6th ed., pp. 864–933). London: HarperCollins Publishers Limited.

Howland, K., & Good, J. (2015). Learning to communicate computationally with flip: A bi-modal programming language for game creation. *Computers & Education*, 80, 224–240. DOI:10.1016/j.compedu.2014.08.014

Hsu, Y. C., & Ching, Y. H. (2013). Mobile app design for teaching and learning educator's experiences in an online graduate course. *The International Review of Research in Open and Distributed Learning*, 14(4)

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52, 200–210. DOI:10.1016/j.chb.2015.05.047

Karakus, M., Uludag, S., Guler, E., Turner, S. W., & Ugur, A. (2012). Teaching computing and programming fundamentals via app inventor for android DOI:10.1109/ITHET.2012.6246020

Kumar, B., & Khurana, P. (2012). Gamification in education – learn computer programming with fun. *International Journal of Computers and Distributed Systems*, 2(1), 46–53.

Lau, W. W. E., & Yuen, A. H. K. (2009). Exploring the effects of gender and learning styles on computer programming performance: Implications for programming pedagogy. *British Journal of Educational Technology*, 40(4), 696–712.

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, 21(1), 57–80.

Margulieux, L., Guzdial, M., & Catrambone, R. (2012). Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. *Proceedings of the Ninth Annual International Conference on International Computing Education Research – ICER'12*, 71–78.

Mladenović, M., Boljat, I., & Žanko, Ž. (2017). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, 1–18. DOI: 10.1007/s10639-017-9673-3

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with scratch programming environment. *Procedia – Social and Behavioral Sciences*, 191, 1479–1482. DOI:10.1016/j.sbspro.2015.04.224

Papert, S. (1993). *Mindstorms: Children, Computers and Powerful Ideas* (2nd ed.) Basic Books.

Prensky, M. (2001). Digital natives, digital immigrants part 1. *On the Horizon*, 9(5), 1–6.

Preston, D. (2006). Using collaborative learning research to enhance pair programming pedagogy. *ACM SIGCSE Newsletter*, 3(1), 16–21.

Robertson, S. A., Lee, M. P., & Miller, J. (1995). The application of second natural language acquisition pedagogy to the teaching of programming languages – a research agenda. *ACM SIGCSE Bulletin*, 27(4), 9–12.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172.

Sáez-López, J., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. *Computers & Education*, 97, 129–141. DOI:10.1016/j.compedu.2016.03.003

Salen, K., & Zimmerman, E. (2003). *Rules of Play: Game Design Fundamentals*. USA: MIT Press.

Sentance, S., & Csizmadia, A. (2015). Teachers' perspectives on successful strategies for teaching computing in school. Paper Presented at IFIP TCS,

Shapiro, R. B., & Ahrens, M. (2016). Beyond blocks: Syntax and semantics. 59(5), 39.

Sprankle, M., & Hubbard, J. (2012). *Problem Solving & Programming Concepts* (9th ed.) Pearson.

Thomsen, B. (2008). Using on-line tutorials in introductory IT courses. In J. Bennedsen, M. E. Caspersen & M. Kölling (Eds.), *Reflections on the Teaching Of Programming* (pp. 68–74). New York: Springer.

Van Gorp, M. J., & Grissom, S. (2001). An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education*, 11(3), 247–260.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. DOI:10.1145/1118178.1118215

Winslow, L. E. (1996). Programming pedagogy – a psychological overview. *SIGCSE Bulletin*, 28(3), 17–25.

Wolber, D., Anderson, H., Spertus, E., & Looney, L. (2014). App inventor 2: Create your own android apps

L. Attard received the B.Ed and M.Ed degrees in Computing Education from the University of Malta in 2012 and 2018 respectively. She is currently a lecturer of Computing at St Aloysius College Sixth Form and a part-time lecturer at the Department of Technology and Entrepreneurship Education at the University of Malta. Her current research and teaching interests include strategies to introduce programming and educational technology.

L. Busuttil is a senior lecturer in Computing Education at the Department of Technology & Entrepreneurship Education (TEE) within the Faculty of Education. He received a PhD from the University of Sheffield in 2014. As part of his work in TEE, Leonard is involved in the formation of pre-service and in-service Computing educators. His research interests include Computational Thinking, Computing education, design of educational software, game-based learning and human computer interaction.