# Perceptions of Scratch Programming among Secondary School Students in KwaZulu-Natal, South Africa

**Mudaray Marimuthu**
*Lecturer, School of Management, IT and Governance, University of KwaZulu-Natal, Westville, Durban*

**Predhayen Govender**
*Honours Graduate, School of Management, IT and Governance, University of KwaZulu-Natal, Westville, Durban*

## Abstract
Scratch programming was designed with the aim of helping students to develop their logical thinking skills as well as enhancing their problem-solving capabilities, without having the technical distractions associated with more advanced programming languages such as Java. This study, guided by the technology acceptance model (TAM), focused on exploring the associations between perceived usefulness, perceived ease of use, attitude towards use, and behavioural intention to use the Scratch programming language, with the aim of identifying how Scratch programming was perceived by a group of South African students in Grades 10 and 11 at two high schools. Results indicated, among other things, that Grade 10 students perceived Scratch to be easy to use and useful, and Grade 11 students found it to be easy to use but useful only in learning introductory programming concepts. These and other findings suggest that while Scratch helps students understand logic and problem-solving, it does not assist sufficiently in preparing them for using a higher-level programming language such as Java. The article concludes with recommendations for South African education policymakers, including proposals that a bridging programming language be introduced between Scratch and Java, and that Scratch be introduced much earlier than in Grade 10.

**Keywords**
Scratch, Java, secondary school students, technology acceptance model (TAM), programming language adoption, visual programming, logical thinking, problem-solving, education, education policy, curriculum, KwaZulu-Natal, South Africa

**DOI:** https://doi.org/10.23962/10539/26112

**Recommended citation**
Marimuthu, M. & Govender, P. (2018). Perceptions of Scratch programming among secondary school students in KwaZulu-Natal, South Africa. *The African Journal of Information and Communication (AJIC), 21,* 51–80.
https://doi.org/10.23962/10539/ 26112

## 1. Introduction

Computer programming is recognised as a vital competence for establishing problem-solving abilities as well as logical and analytical reasoning. Its integration throughout various educational levels is regarded as valuable, with many studies carried out to further explore this phenomenon (see, for example, Annamalai & Salam, 2017; Fessakis, Gouli, & Mavroudi, 2013; Isa & Derus, 2017; Meyerovich & Rabkin, 2013; Ozmen & Alten, 2014; Tom, 2015).

Studies within the field of computer science have revealed a lack of problem-solving and computational skills among students in introductory programming courses, despite programming becoming so important in the 21st century (Papadopoulos & Tegos, 2012; Tan, Ting & Ling, 2009; Robins, Rountree & Rountree, 2003).

The Scratch programming language was designed and developed at the Massachusetts Institute of Technology (MIT) Media Lab with the intention of simplifying the process of developing and programming animations, interactive stories, and games. Lamb and Johnson (2011) state that, with respect to computer software, the term "scratching" refers to reusable code segments that can be instrumentally and functionally adapted to new scenarios and used for other purposes. The word "scratch" is said to be derived from the turntablism method of scratching, with the Scratch programming platform associating the technique of mixing sounds to the mixing of software projects. Through Scratch programming, users can upload web-based or downloaded software projects to the website for sharing purposes. Credit is awarded to the participants who develop the initial programs.

Scratch is primarily focused on children and teenagers, with the intention of conveying computational thinking using a simple but cogent building-block approach in the software development process, focused less on programming detail than on emphasising the problem-solving aspects (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Scratch offers young individuals the freedom to “imagine, program and share” (Housand & Housand, 2011, p. 22). Scratch does not require any programming knowledge, and has an intuitive interface, noted as a necessity for its young audience. With Scratch, users are able to build scripts by selecting blocks of code that govern motion, color and sensors. These scripts define specific operations with respect to the program’s objects. The building blocks of Scratch programming make it easy for users to piece together the necessary programming elements without programming knowledge (Watters, 2011).

Since the release of Scratch in 2007, more than 850,000 users have joined the Scratch website and have shared almost two million projects, many of which are animations or games. The ability for users to easily share information has become a fundamental feature of the Scratch platform. Uploaded projects are licensed under the Creative Commons Attribution Share Alike licence. This entails users freely downloading graphics and source code from online projects and reusing components from them with minimal constraint (Watters, 2011).

According to Meyerovich and Rabkin (2013), programming languages such as Java, C and C# are widely popular. Understanding the factors of a successfully adopted language can help inform efforts by advocates and language designers to influence the languages' comprehensive function and design. Educators often encounter problems with the teaching processes associated with programming logic and skills, which  calls into question teaching methodologies. Studies have shown that many students lack problem-solving and computational thinking abilities (Papadopoulos & Tegos, 2012) and these skills have been identified as important competencies in the 21st century (Marques & Marques, 2012).

Education authorities in South Africa want Grade 10 students to learn basic programming principles and constructs with a fun and easy-to-learn tool. Therefore Scratch has been implemented in schools to introduce students to "important computational skills and concepts, algorithm development, problem solving and programming"  (Department of Basic Education, 2011, p. 12). Our study aimed to measure the acceptance of Scratch programming by Grade 10 and 11 students by analysing whether perceived usefulness, attitude to use and perceived ease of use influence students' behavioural intention to use Scratch, which in turn can be expected to influence its acceptance. The framework for the study was grounded in the technology acceptance model (TAM).

The research questions underpinning the study were:

1) To what extent does perceived usefulness influence a student's behavioural intention to use Scratch programming?
2) To what extent does perceived ease of use influence a student's behavioural intention to use Scratch programming?
3) To what extend does the attitude towards using Scratch influence a student's behavioural intention to use Scratch?
4) Are there differences in perceptions between Grade 10 and Grade 11 students in respect of perceived usefulness, perceived ease of use, attitude towards using, and behavioural intention to use Scratch programming?
5) Does teaching Scratch in Grade 10 make it easier for learners in Grade 11 to learn Java?

## 2. Literature review and theoretical framework

### *Problems with learning programming*

Factors that have been found to affect teaching programming to young individuals include applying programming concepts to situations involving complex  problems, syntax complexity, and associating programming with tasks unrelated to the interests or thought processes of young individuals (Maloney et al., 2010). Papadopoulos and Tegos (2012) similarly state that studies within the field of computer science have shown that students lack problem-solving as well as logical thinking abilities. Robins et al. (2003) identify the most crucial shortcomings with students learning programming as being associated with problem-solving activities, developing and designing solutions, and expressing the designed solutions as programs.

Many programming languages are difficult to comprehend to the untrained eye, due to a mixture of English and unfathomable programming language syntax. Programming syntax is the set of rules and symbols of a programming language, which enable a programmer to create correctly-structured programs. The sheer magnitude of syntax and keywords in a basic Java program would defy explanation on the first day of an introductory programming class. Although the "mastery of precision" can be seen as fundamental when learning programming (Malan & Leitner, 2007), in the early stages of an introductory course it can often be found that semicolons, parentheses and other syntactical elements delay and discourage students from understanding significant programmatic constructs such as variables, conditionals, loops or even the logic itself (Malan & Leitner, 2007). Many programming languages, Java included, compel students to grasp the programmatic overheads before actually programming.

Several studies have found that students encounter difficulties with the initial steps of programming. A study by McCracken et al. (2001) found that students in their first one or two courses in computer science experienced difficulties with the reading, writing and designing of code. Tan et al. (2009) conducted a survey to determine the possible factors that lead to problems with learning programming. Taking into consideration students' computing experience and background, Tan et al. (2009) concluded that the majority of students encountered problems with memory-related concepts such as the storage and manipulation of variables in the computer's main memory. This finding concurs with that of Milne and Rowe (2002), who found that many students were incapable of developing a simple "mental model of memory movement" during the execution of the program. Beginner programmers lack clear mental models and fail to apply the relevant knowledge. They focus more on little problems rather than on the planning and testing of code (Milner & Rowe, 2002).

According to Rudder, Bernard and Mohammed (2007), an effective method for students to learn programming is to translate real-world problems into code, and solve them accordingly. However students have found this method to be difficult, since daily real-world situations are in a single context while the task of learning programming is considered a multilayered skill. Multilayered skills are abstract since programming languages are designed for the unfamiliar realm of computers rather than the natural world experienced by people when growing up (Moser, 1997).

Ozmen and Alten (2014), and Bosse and Gerosa (2017), indicate that many problems with students learning computer programming stem from the complexity of programmatic constructs such as programming syntax, variables, functions, and loops. Complexities such as these may be seen as a barrier for students to learn programming and may even hinder their motivation to learn.

***Effectiveness of using programming and visual tools***

Due to the rapid growth of digital technology "individuals are required to use a growing variety of technical, cognitive, and sociological skills in order to perform tasks and solve problems in digital environments" (Eshet-Alkalai, 2004, p. 93). To assist in the attaining of these skills, teaching students to program has been introduced into primary and secondary education curriculums.

Pendergast (2006) states that the significance of a well-constructed introduction to programming course cannot be over-emphasised, as it was observed that many students found difficulties with understanding the programming process as well as familiarising themselves with the various programming constructs. Visual programming tools like Alice, a programming language developed by Carnegie Mellon University, and Scratch, are favoured with younger introductory students (Lye, & Koh, 2014). Alice, which was developed before Scratch, is used to teach students general programming concepts as well as object-oriented programming (OOP) concepts. Much effort has gone into the development of visual programming tools for young individuals (Meerbaum-Salant, Armoni, & Ben-Ari, 2013), with these tools being used by young children and as a preliminary learning tool for secondary schools and universities. Visual programming tools such as these create a non-threatening, fun environment for students to develop software, in a way that aims to reduce the anxiety and fear often associated with learning programming (Meerbaum-Salant et al., 2013). It is believed that through these environments, students will be more open to continuing their study of programming.

A study by Boyle, Bradley, Chalk, Jones and Pickard (2003) focused specifically on a "visual approach", making use of graphical shapes to teach abstract programming concepts which were available to students in a virtual learning environment (VLE). Boyle et al. (2003) implemented this approach with an introduction to programming course, and found a 12% to 23% increase in the pass rate over the previous year's students who did not have a VLE- and a graphics-based approach. The results of a questionnaire handed to students during mid-semester showed that 95% of students judged the graphics-based approach to learning programming to be "good" or "very good" (Boyle et al., 2003).

Prior to the development of Scratch, there were many other programming environments, such as Alice, Logo and Karel, all attempting to make learning programming simpler for the beginner programmer. Alice, Karel and Scratch are all visual programming environments.  If Alice, Logo and Karel have one weakness, it is their steeper learning curve than that of Scratch (Malan & Leitner, 2007). Programming tools such as Logo have been viewed as an opportunity for students to expand their intellectual capabilities to take on challenging problems (Papert, 1980). However, it has been noted that programming languages such as Logo have not flourished as expected (Mannila, Peltomaki, & Salakoski, 2006). According to Lee (2011), many of the difficulties encountered by students when using Logo can be attributed to the limitations of the programming tool. One particular example stems from syntactic constraints. In Logo, every line of code must comply with the syntactic constraints (the programming language rules) before the program can be tested. A fragile environment such as this could see students paying more attention to the syntax of the program, while less emphasis is placed on the semantic meaning. Lee (2011) found that

> One of the reasons for the low adoption of computer programming in K–12 education is the time it takes for (especially young) students to learn computer programming using a textbased programming language, which requires an understanding of computer programming language syntaxes and constructs and strong keyboarding skills. (Lee, 2011, p. 27)

Alice is a free interactive 3D-programming tool to help students gain exposure to OOP concepts (Ebrahimi, Geranzeli, & Shokouhi, 2013). With Alice, students are able to learn fundamental programming concepts in the form of creating video games or animations in a visual programming environment, unlike Logo and Karel where the environment in text-based, thereby creating a steeper learning curve.

In recent years, Scratch and Alice have both been used at university level in introductory computer science courses. Lewis (2010) found that new languages like Scratch are often developed and modeled from existing languages, to provide new functionality while offering claimed pedagogical advantages. Scratch reduces the syntax complexity of Alice, which has class-based OOP and emphasises Java or Java-related concepts (Maloney et al., 2010). Lewis (2010) conducted a study which built upon existing research, aimed at testing the pedagogical claims of new programming environments. The study assessed Scratch's pedagogical value in contrast with Logo. Lewis (2010) hypothesised that students who learnt programming using Scratch, as opposed to using Logo, would be more confident about their skills as programmers, would be more capable of tracing the flow of control of conditions and loops, and would report that learning programming concepts in general was easier. The study found that when interpreting loops, students learning through Scratch and Logo performed similarly, regardless of the fact that Logo was textual while Scratch was visual. The Logo environment was thus able to support the development of confidence in students when learning programming as well as spike their interest in the field. However, students using Scratch performed better than the Logo students when interpreting conditionals, and Scratch made general programming concepts easier to interpret and learn.

Parsons and Haden (2007) conducted a study to test whether students currently learning Java would find Alice useful for developing programming competencies with flow-of-control constructs. It was concluded that students struggled to make the connection between work in Alice and "real programming". In an attempt to reduce syntactic constraints, many visual programming languages, Alice included, may be perceived as too "simple" and not related to "real programming" (Lewis, 2010).

***Scratch programming***

Maloney et al. (2010) define the Scratch programming language as a programming domain allowing individuals, predominantly between the ages of 9 and 17, to learn and understand fundamental programming concepts while also being able to develop purposefully meaningful projects like games or animations. Certain experts in the field are devoted to trying to find feasible and interesting ways of reviving the primary objective of making programming accessible and interesting to young individuals. It was on this basis that the Scratch programming platform was conceived and developed.

Marques and Marques (2012) place emphasis on significant competencies such as problem-solving and critical thinking skills, which are essential for the 21st century. They further state that through the use of Scratch, users are able to positively develop these competencies.

Kim, Choi, Han, and So (2011) designed a computer course for student educators using Scratch, owing to the fact that student educators encountered difficulties mastering programming language syntax. The course was developed with the intention of encouraging computational skills as well as promoting creative thinking. Kim et al. (2011) conclude that Scratch helped student educators grasp fundamental programming concepts and focus implicitly on what they were able to do with Scratch.

Theodorou and Kordaki (2010) used Scratch to design and develop a computer game with the goal of providing high school learners with a learning habitat to promote various programming concepts. It was concluded that Scratch was a very useful environment because the programming is "done by constructing blocks of simple commands and not by writing text commands" (Theodorou & Kordaki, 2010, p. 13). Topalli and Cagiltay (2018) augmented their course curriculum by including game development projects using the Scratch environment. The findings show that Scratch helped learners perform better in introductory programming courses.

The capabilities of the Scratch learning environment are also emphasised by Lee (2011), who suggests that teachers can benefit from the Scratch platform by developing more entertaining and creative materials so that students can free their imaginations in a consequential manner.

Lai and Yang (2011) conducted a study to assess the effect of visual programming, using Scratch, on students' logical and problem-solving abilities. Lai and Yang (2011) posit that problem-solving abilities include "grasping the problem, analysing the problem, finding out solutions and writing program, verifying the solution by testing, and modifying the program according to the result of the test" (p. 6941). They sampled Grade 6 students who had taken a Scratch programming course, and noted a distinct improvement in the students' problem-solving abilities. Hence, it was concluded that visual programming could enhance problem-solving. Similar studies by Calao, Moreno-Leon, Correa and Robles (2015) and Calder (2010) also show that Scratch increases the logical thinking and problem-solving abilities of young students. Korkmaz (2016) found that an educational programme based on Scratch-related game activities resulted in a significant positive contribution to the logical-mathematical thinking skills of students, more so than educational programmes using Lego Mindstorms Ev3 design activities or traditional teaching activities.

A study by Wilson and Moffat (2010) assessed student use of Scratch programming in an Information Technology module for a period of eight weeks. This module was taken by primary school pupils between the ages of 8 and 9. The researchers concentrated on two distinct aspects: whether various programming concepts were efficiently conveyed via Scratch programming (cognitive), and whether it was fun and easy to use (affective). The results showed a moderate increase in student performance and a more enjoyable experience for students, making learning to program a positive experience. The researchers concluded that, for an ideal educational system to implement successful teaching of programming and programming concepts, primary focus should be not only on a student's cognitive dimension but just as importantly the student's emotional state. Kalelioğlu and Gülbahar (2014) claim that Grade 5 students in their study also found the Scratch platform easy to use. It was clear from the study that Scratch does undoubtedly excite. Studies by Permatasari, Yuana and Maryono (2018) and Sáez-López, Román-González and Vázquez-Cano (2016) also demonstrated that students found Scratch to be easy and fun, made them enthusiastic and motivated about learning programming, and even motivated them to continue studies in programming.

A similar study conducted by Baytak and Land (2011) explored the development process employed by Grade 5 students to design and build computer-based games using the Scratch programming language utilising a "learning by doing" approach. It was concluded that students were more likely to enhance their programming abilities and create computer games when the visual-programming software they employed was suited for their level of experience. Iskrenovic-Momcilovic (2017) demonstrated that by not having syntax complexity, Scratch allowed beginner programmers to solve complex problems quickly.

*Scratch programming in the South African context*
In 2011, the South African Department of Basic Education introduced its Curriculum and Assessment Policy Statement (CAPS). In terms of CAPS, educators were mandated to introduce Scratch to Grade 10 learners, as a gateway to learning other programming languages such as Java. There is still only limited research (Beyers and Van der Merwe, 2017; Van Zyl, Mentz & Havenga, 2016; Koorsse, Cilliers, Calitz, 2015; Chetty & Barlow-Jones, 2012) on Scratch programming in the South African context and on South African students' perceptions of Scratch and similar tools.

*Scratch programming as a platform for introducing Java*
According to Malan and Leitner (2007), Scratch can be seen as a viable gateway to programming languages such as Java or Python. Research by Malan and Leitner (2007) found that Scratch not only thrills and excites students in the early stages of programming but also exposes inexperienced students to fundamental programmatic concepts without the "distraction of syntax". Basic fundamental programming concepts are sequence (doing things in the correct order to solve a problem), decisions (e.g., using "if" conditions to execute instructions based on a true or false decision), and repetition (using loops for execution of instructions more than once). Malan and Leitner (2007) found that while Scratch does not support complex programming constructs such as methods, data types, and parameters, many of which could be considered crucial in an introductory course, its simplicity and power to allow inexperienced students to learn fundamentals programming concepts are what keep students engaged and excited. Students, justifying the time spent working on the program, stated that "Scratch is fun to use and really easy to learn, almost addictive in a way" (Malan & Leitner, 2007, p. 5).

However, in the same Malan and Leitner (2007) study, some students stated that they found Scratch negatively influenced their preparation to take on Java. One respondent stated that Scratch was easy and a lot of fun, but not good enough preparation for the jump to the  good preparation for Java. Another student comment was:

> I think Scratch didn't really help me with Java. I had fun with Scratch and I see how it could serve as a didactic tool for some people but I would have preferred to jump straight into Java. The elements of programming that Scratch attempts to teach are not particularly difficult to understand and I feel may be 'safely' introduced using Java itself, I feel we could have progressed a lot more into Java had we jumped directly into it. (Malan & Leitner, 2007, p. 6)

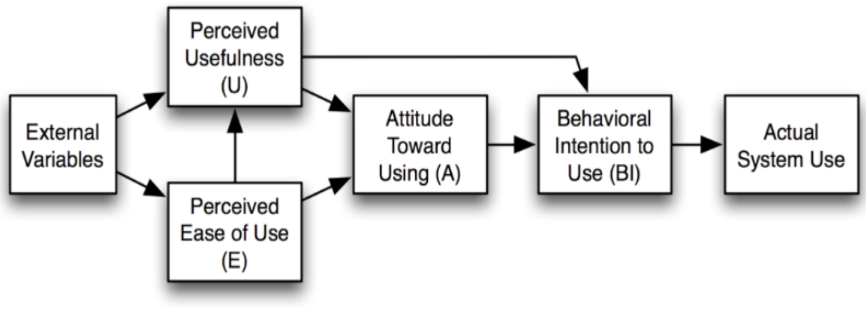
## *Theoretical framework*

The technology acceptance model (TAM) (Davis, Bagozzi & Warshaw, 1989) has been employed, due to its predictive ability, in many studies involving learners (Ibrahim, Leng, Yusoff, Samy, Masrom & Rizman, 2017; Sánchez-Mena, Martí-Parreño & Aldás-Manzano, 2017; Mugo, Njagi, Chemwei & Motanya, 2017; Olivier, 2016; Cakir & Solak, 2014). According to Davis et al. (1989), perceived usefulness (U) is a cognitive evaluation of whether the adoption of a new technology will impact an individual's job performance. Perceived usefulness influences an individual's attitude towards use (A) of new technologies, due to the fact that people tend to form favourable attitudes towards new technologies through the belief that the technologies will impact their job performance in a constructive manner. Perceived usefulness also directly affects the behavioural intention to use (BI) the system. This is based on the idea that, regardless of their personal feelings for the technology, individuals develop intentions to use a device with the belief that it will positively affect job performance, because people are inspired to obtain performance-contingent rewards such as raises or promotions (Davis et al., 1989; Davis, 1989).

Perceived ease of use (E) is found to impact both attitude towards using and perceived usefulness. Self-efficacy of a user is likely to be impacted when a system is easier. An individual with high self-efficacy for a system has strong belief in their ability to use the system, resulting in a more enthusiastic attitude towards the system. Ease of use also directly affects an individual's performance, since the new technology is likely to lead to completion of a task using less effort (Davis et al., 1989; Davis, 1989).

For our study, perceived usefulness was conceptualised as the extent to which a student believes that utilising the Scratch programming language would enhance their programming ability and overall performance in the course. We used perceived ease of use to refer to the degree to which an individual believes that utilising the Scratch programming platform will be free of cognitive effort. Viewed in terms of the TAM framework, the actual usage of Scratch may be determined by the user's behavioural intention to use Scratch, which in turn is determined by the user's overall attitude towards Scratch as well as her or his perception of usefulness and ease of use, i.e., according to Davis et al. (1989), perceived usefulness together with perceived ease of use have a significant influence on attitude, which in turn impacts behavioural intention to use.

**Figure 1: TAM, as set out by Davis et al. (1989)**


Perceived Usefulness (U)
External Variables
Perceived Ease of Use (E)
Attitude Toward Using (A)
Behavioral Intention to Use (BI)
Actual System Use

**Source: Davis, Bagozzi, and Warshaw (1989, p. 985)**

## 3. Methodology

### *Research approach*

The study used a mixed-method approach, encompassing both qualitative and quantitative data collection and analysis techniques. The mixed-method approach was useful because it allowed the open-ended questions (qualitative) to provide insight and understanding to the quantitative data collected.

Quantitative research focuses on the numbers behind a survey and uses statistics to generalise findings. Quantitative data were used to analyse the associations between the four constructs of TAM—i.e., perceived usefulness, perceived ease of use, attitude towards use, and behavioural intention to use—in order to address the research questions.

### *Target population and sampling*

The target population for this study was 70 Grade 10 and Grade 11 Information Technology students from two secondary schools in South Africa's KwaZulu-Natal Province. The Grade 10 students were learning programming using Scratch, while the Grade 11 students had already made the transition from Scratch to Java. Due to the small population size, all 70 students were targeted for this study.

### *The questionnaire*

The questionnaire consisted of both open-ended and closed-ended questions. Closed-ended questions utilised a 5-point Likert scale to obtain an understanding about students' opinions on the effectiveness of Scratch. The open-ended questions were used to get a better understanding of Grade 11 students' perception of Scratch after programming in Java.

### *Data collection*

After receiving ethical approval from the University of KwaZulu-Natal, the questionnaire, along with an accompanying letter of consent, was manually issued to respondents. Letters of consent were also issued to parents/guardians, due to the fact that all respondents were below the age of 18. Consent letters were signed by research participants and also by participants' parents. Respondents were also made aware that the data received from them would be anonymous.

### *Data analysis*

To ensure reliability of data, Cronbach's alpha was calculated among the Likert scale questions of the questionnaire. Cronbach's alpha is a measure of internal consistency, used to determine how closely related a collection of values are as a group. A Spearman correlation coefficient was used to measure the strength and direction between the four constructs of the technology acceptance model. Spearman correlation was chosen since the quantitative data obtained were nominal (ranked on a Likert scale) and not normally distributed (Chok, 2010). Also, the Mann-Whitney test was used to determine if there were significant differences in perceptions between Grade 10 and Grade 11 students.

## 4. Findings and discussion

The quantitative data and qualitative data obtained from the questionnaire were analysed to identify and explore the relationships between four variables in respect of use of Scratch—perceived usefulness (U), perceived ease of use (E), attitude towards using (A), and behavioral intention to use (BI)—as specified by TAM.

An initial target size of 70 was planned for the study. Ultimately 47 surveys were returned, and 45 could be analysed. The 45 respondents consisted of 23 Grade 10 students and 22 Grade 11 students, which amounted to 64% of the total population. (Two of the completed questionnaires were deemed unusable for the study as many questions were unanswered.) Descriptive statistical analysis and correlation analysis techniques were utilised to analyse percentages and frequencies of the Likert scale questions in the questionnaire. Reliability analysis was used to measure reliability of the data.

### *Reliability analysis*

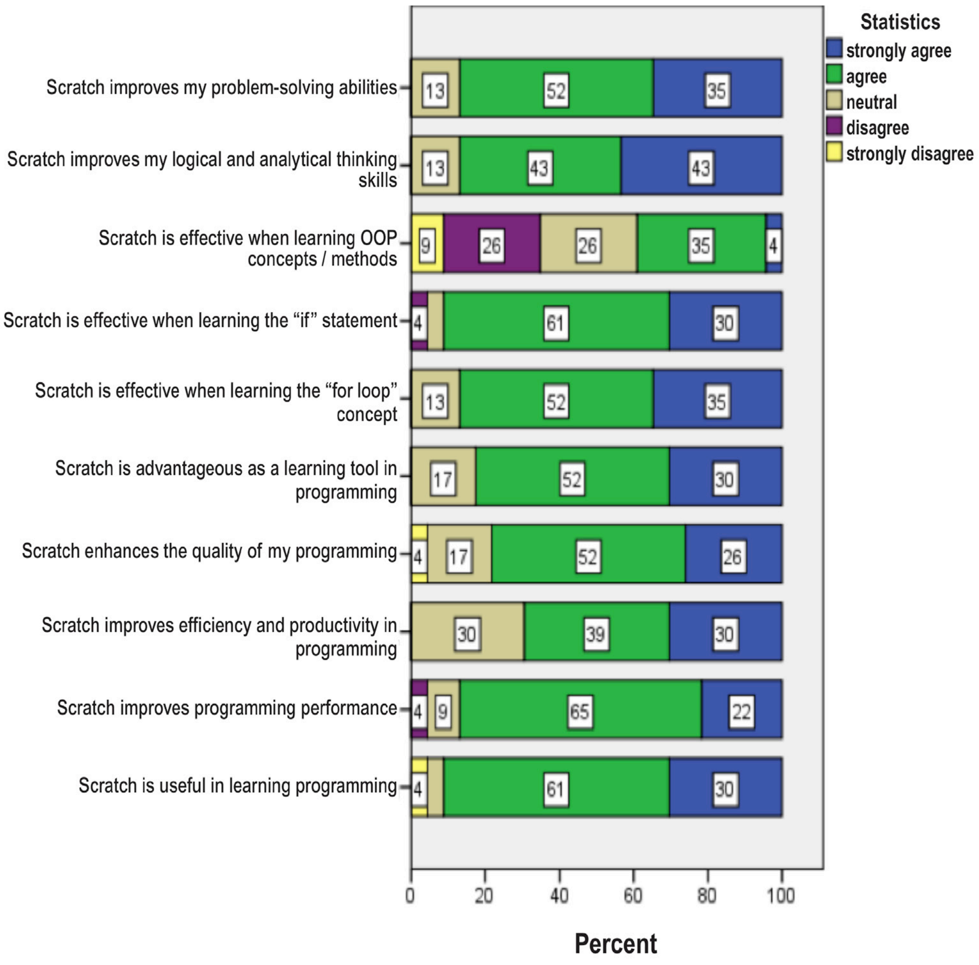
As stated above, to test reliability of the data, Cronbach's alpha was calculated among the Likert scale questions of the questionnaire. The questions were grouped into four constructs of TAM, namely U, E, A and BI. A reliability coefficient of 0.7 or higher is generally considered acceptable (Tavakol & Dennick, 2011). The results for all four groups of questions showed values greater than 0.7, indicating that with all four groups, the items within the group had acceptable levels of consistency. The 10 questions within the U factor obtained an alpha coefficient of 0.751. The five questions within the E factor obtained an alpha coefficient of 0.801. The four questions within the A factor obtained an alpha coefficient of 0.864. The three questions within the BI factor obtained an alpha coefficient of 0.867.

## *Descriptive statistical analysis*

*Perceived usefulness*

Figures 2 and 3 illustrate the descriptive statistical findings for perceived usefulness of Scratch.

**Figure 2: Perceived usefulness of Scratch among Grade 10 respondents**



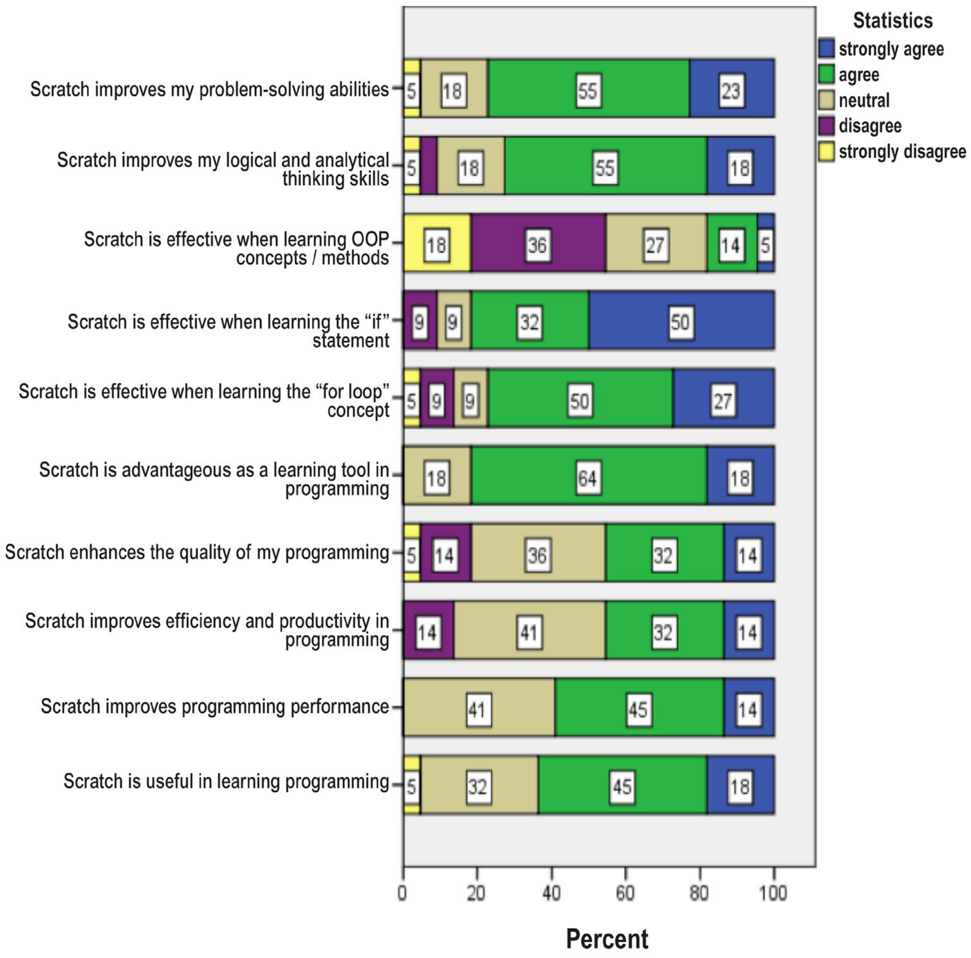
As shown in Figure 2, the statement "Scratch is effective when learning OOP concepts/methods" had a nearly equal percentage of negative and positive responses among the Grade 10 respondents. (OOP, alluded to earlier, is object-oriented programming, in which objects have attributes that are assigned data and these attributes can be manipulated.) For all the other factors, the majority of the Grade 10 respondents agreed or strongly agreed. Of these statements, "Scratch improves efficiency and productivity in programming" received agreement or strong agreement from 69% of the respondents, while the percentage of those who agreed or strongly agreed ranged from 82% to 91% for the remaining statements.

As shown in Figure 3, Grade 11 students perceived the usefulness of Scratch positively in terms of all statements except the statement "Scratch is effective when learning OOP concepts/methods." There was majority neutrality or disagreement in response to this statement among Grade 11 students (27% were neutral, 54% disagreed or strongly disagreed). This was presumably due to Grade 11 students having being exposed to OOP concepts in the Java programming environment and therefore seeing the Scratch programming environment as inferior in terms of learning OOP.

The positive responses were sharply lower among Grade 11 students than among Grade 10 students for the following factors: "Scratch is useful in learning programming", and "Scratch improves programming performance." Thus, it became apparent that after students had been exposed to the highly syntax-based environment of Java, they tended to feel that Scratch was not useful in learning programming and or improving their programming performance.

Both groups of students had similar percentages of positive responses for the factors "Scratch improves my logical and analytical thinking skills" and "Scratch improves my problem-solving abilities." Thus, even after exposure to Java, the Grade 11 students still positively perceived Scratch's ability to improve their logical/analytical thinking skills and problem-solving abilities.

**Figure 3: Perceived usefulness of Scratch among Grade 11 respondents**



*Perceived ease of use*
Figures 4 and 5 illustrate the descriptive statistical findings for perceived ease of use of Scratch.

**Figure 4: Perceived ease of use of Scratch among Grade 10 respondents**

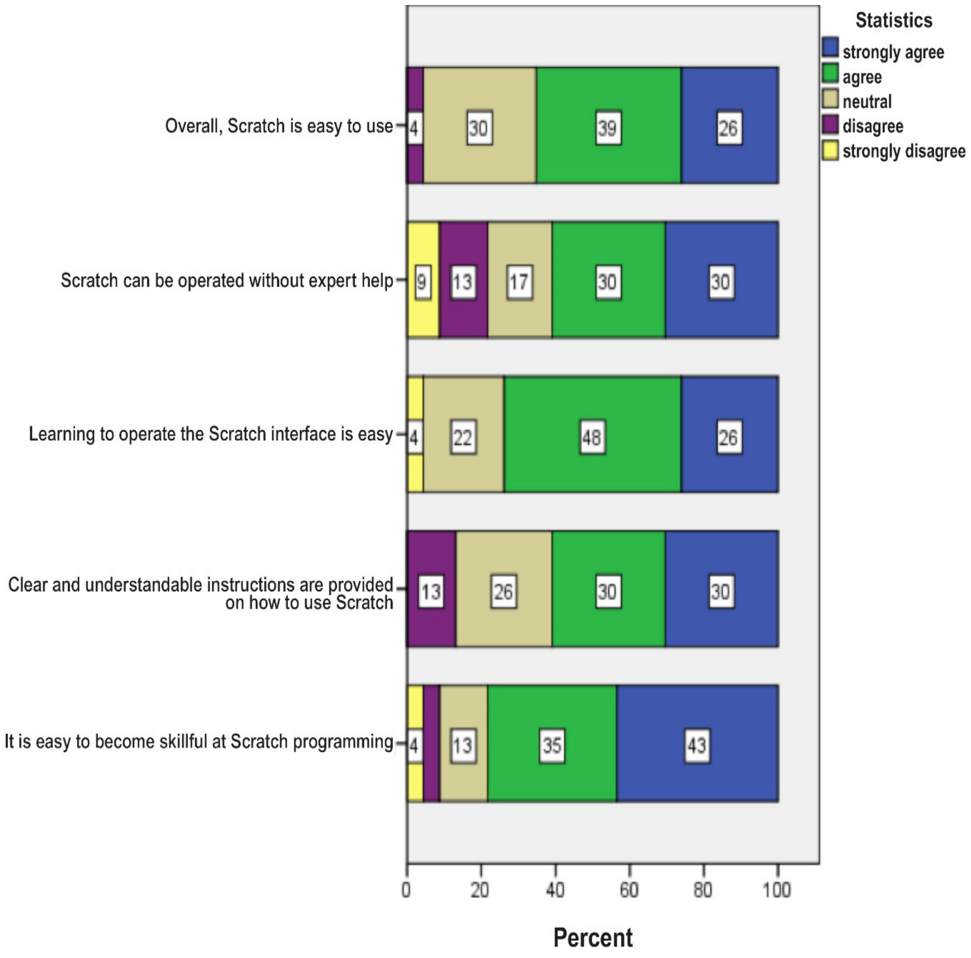
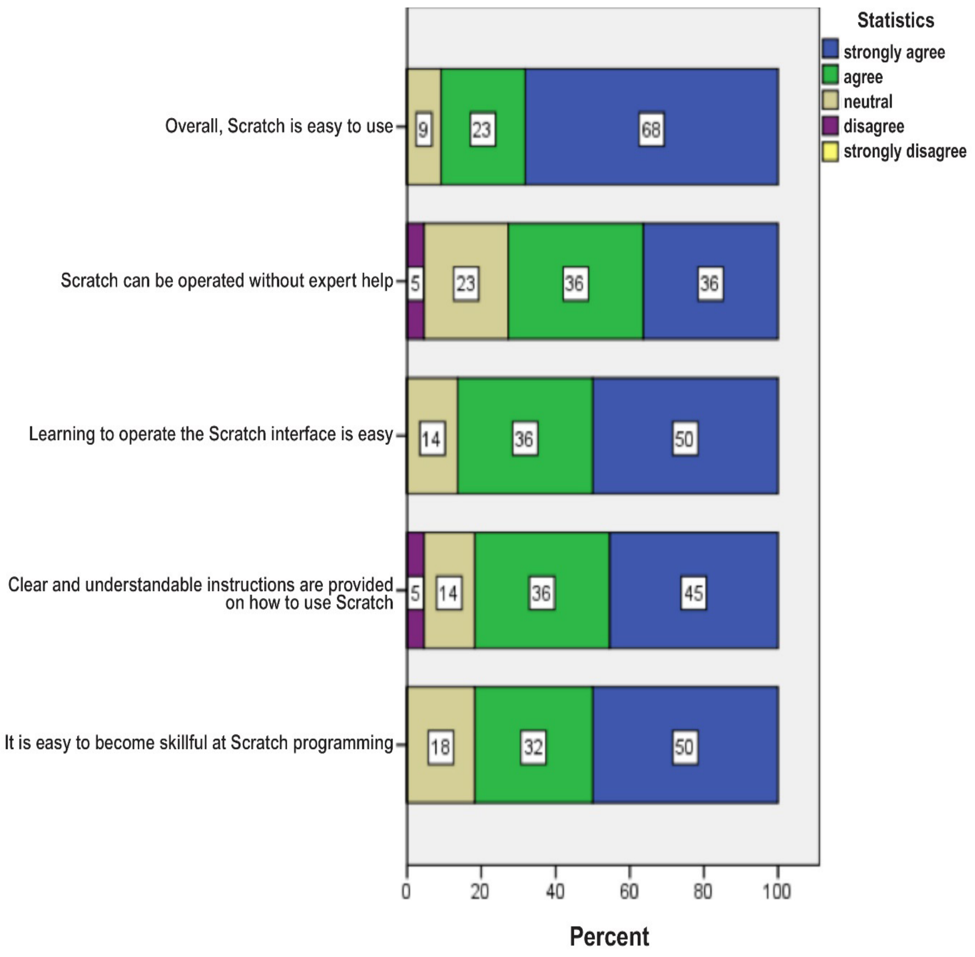


**Figure 5: Perceived ease of use of Scratch among Grade 11 respondents**

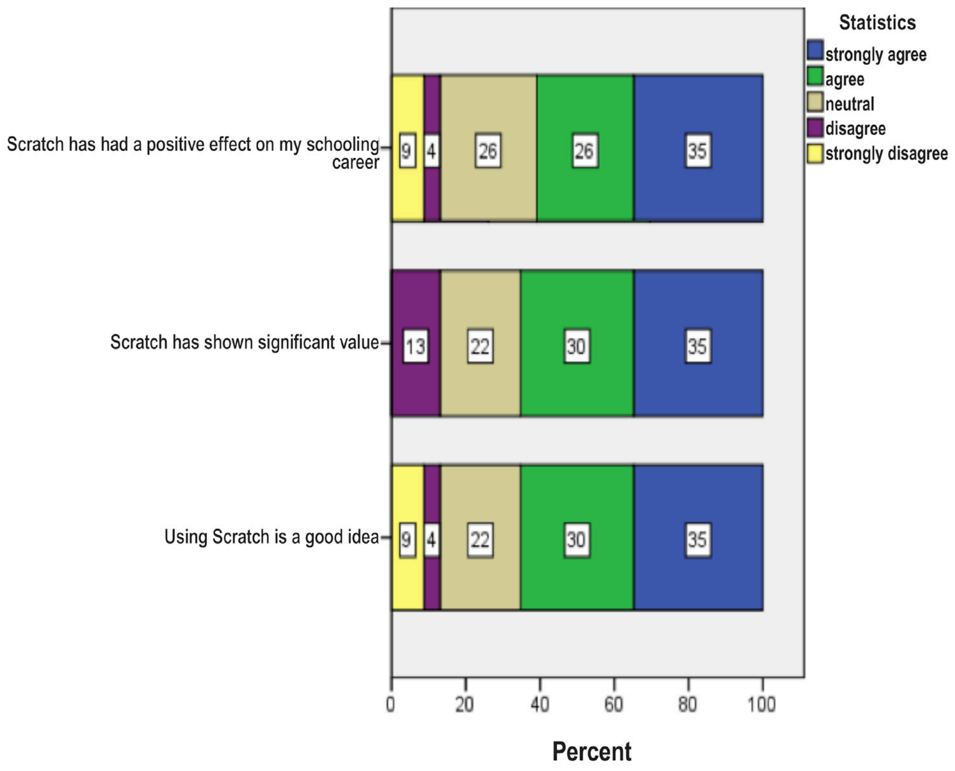


As seen in Figures in 4 and 5, both the Grade 10 and Grade 11 students were found to have mostly positive perceptions (strongly agree or agree) in respect of the ease of use construct, thus indicating that most students found Scratch easy to use. When the two groups of responses are compared, one sees a greater percentage of positive responses from Grade 11 students than Grade 10 students in respect of ease of use. This difference was not unexpected, as the Grade 10 students were being exposed to programming for the first time, and thus it made sense that would find Scratch difficult to use. Meanwhile, the Grade 11 students had now been exposed to the Java programming environment, and could be expected to find the Scratch environment comparatively easy, thus contributing to greater number of positive responses from Grade 11 students compared to the Grade 10 learners.

*Attitude towards using*

Figures 6 and 7 illustrate descriptive statistical findings for respondents' attitude towards using Scratch.

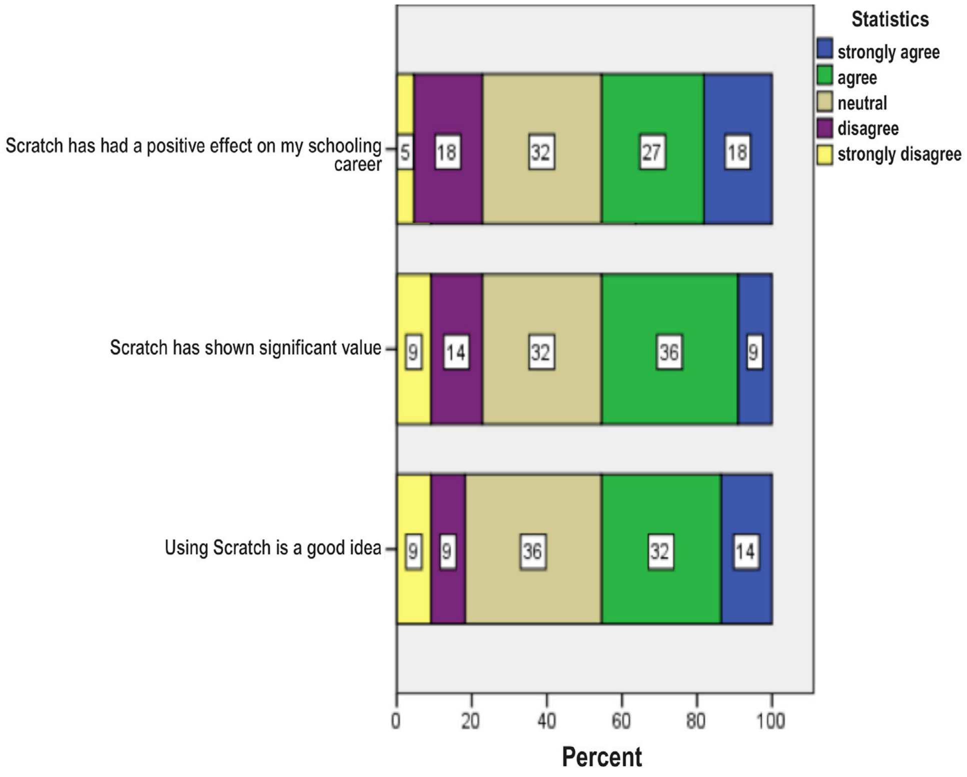
**Figure 6: Attitude towards using Scratch among Grade 10 respondents**



As Figure 6 shows, the majority of Grade 10 students were favourable in their attitude towards using Scratch. Their total positive responses (either agree or strongly agree) ranged from 61% to 65%.

Meanwhile, as seen in Figure 7, for Grade 11 students, favourable responses (strongly agree or agree) were never in the majority, with the three statements receiving fewer than 50% favourable responses, i.e., "Using Scratch is a good idea" (46%), "Scratch has shown significant value" (45%), "Scratch has had a positive effect in my schooling career" (45%). These findings were apparently a result of Grade 11 students' exposure to Java, which made them able to identify the limitations of Scratch.

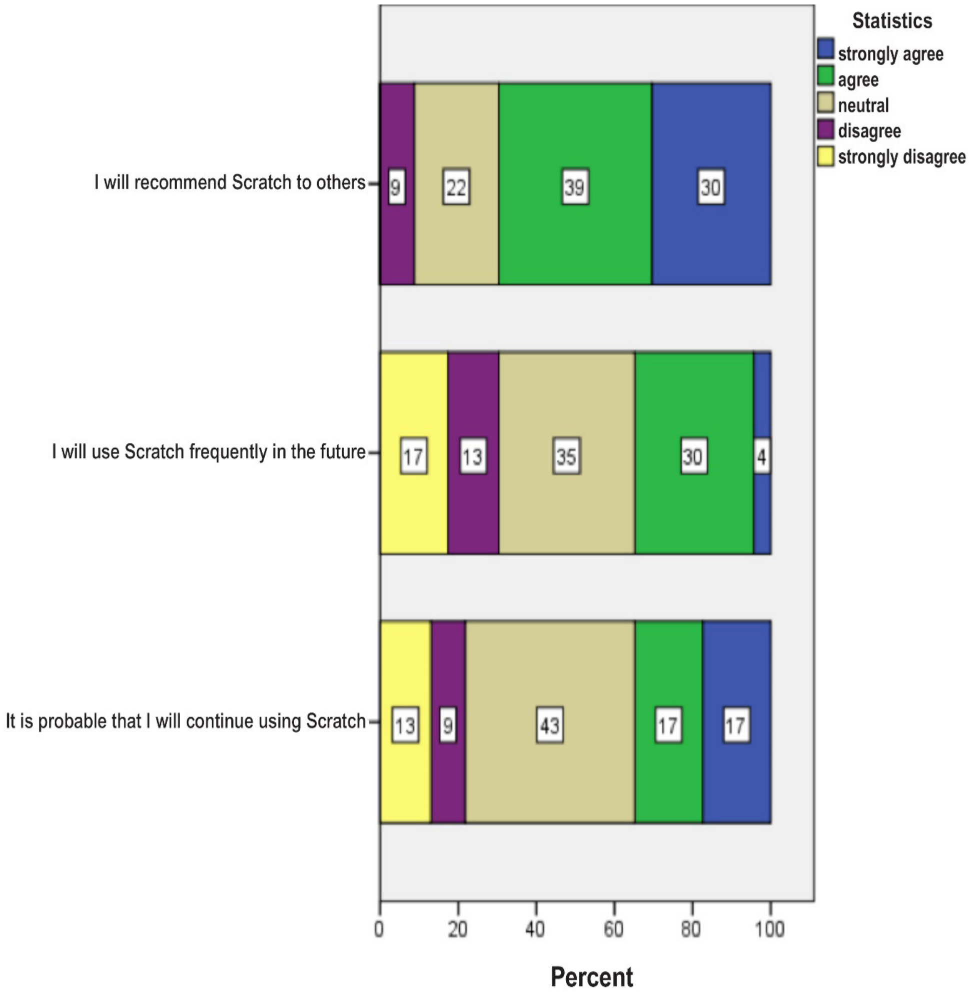
**Figure 7: Attitude towards using Scratch among Grade 11 respondents**



*Behavioural intention to use*

Figures 8 and 9 illustrate the descriptive statistical findings for respondents' behavioural intention to use Scratch.

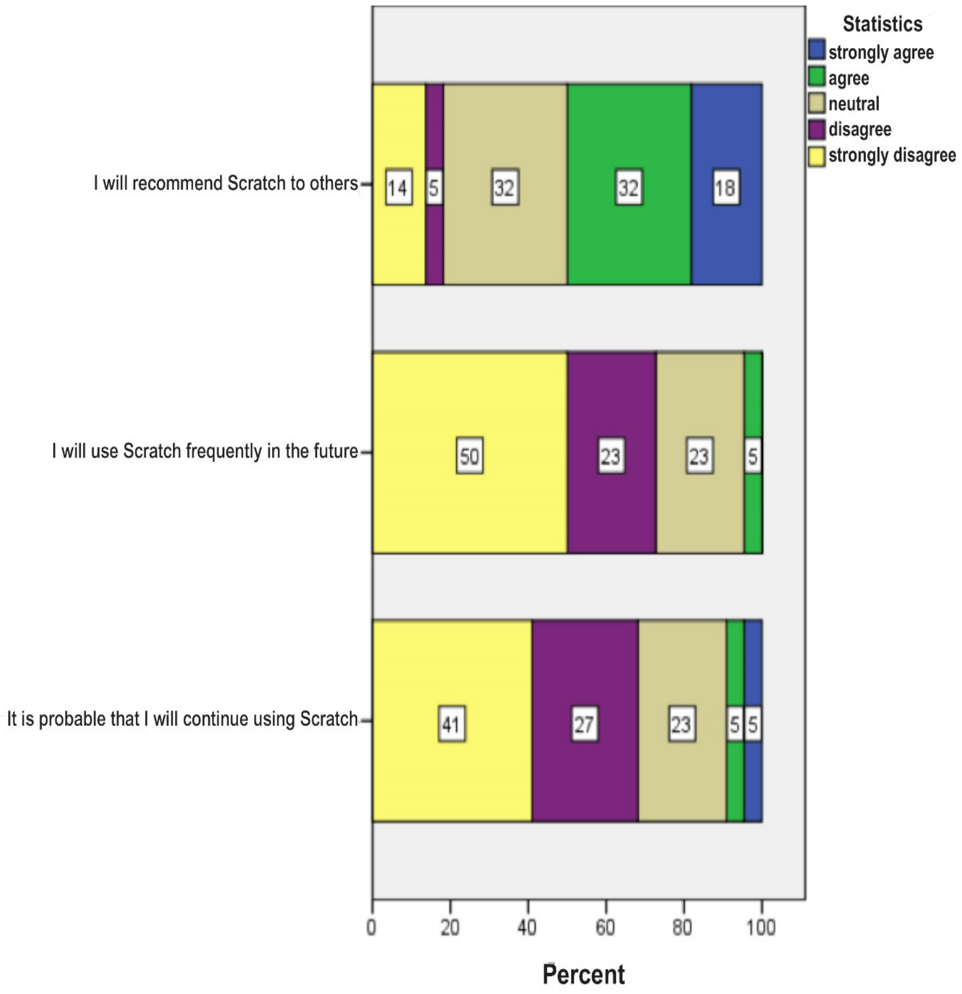
**Figure 8: Behavioural intention to use Scratch among Grade 10 respondents**



As seen in Figure 8, among the Grade 10 students, 69% of respondents either agreed or strongly agreed that they would recommend Scratch to others. However, there were much less favourable responses to the statements "I will use Scratch frequently in the future" (only 34% agreed or strongly agreed) and "It is probable that I will continue using Scratch" (only 34% agreed or strongly agreed).

Meanwhile, for Grade 11 respondents, as shown in Figure 9, 50% agreed or strongly agreed that they would recommend Scratch to others. But for the other two statements, the majority of respondents disagreed or strongly disagreed, i.e., the statements "I will use Scratch frequently in the future" (73% disagreed or strongly disagreed) and "It is probable that I will continue using Scratch" (68% disagreed or strongly disagreed). Thus, this suggests that after exposure to Java, the Grade 11 students had become aware of the limitations in the capabilities of Scratch and, accordingly, perceived that they did not intend to use it going forward.

**Figure 9: Behavioural intention to use Scratch among Grade 11 respondents**



***Mann-Whitney test***

Since the data were not normally distributed, a Mann-Whitney test was performed to determine if there were significant differences in perception between Grade 10 and Grade 11 students with regard to the TAM factors.

**Table 1: Mean values for TAM constructs, Grades 10 and 11**

| TAM Factor | Grade | N | Mean | Std. Deviation | Std. Error Mean |
|---|---|---|---|---|---|
| **U** | Grade 10<br>Grade 11 | 23<br>22 | 1.9826<br>2.3455 | .44889<br>.47180 | .09360<br>.10059 |
| **E** | Grade 10<br>Grade 11 | 23<br>22 | 2.1478<br>1.6909 | .74399<br>.61945 | .15513<br>.13207 |
| **A** | Grade 10<br>Grade 11 | 23<br>22 | 2.2609<br>2.7159 | 1.09357<br>.87728 | .22803<br>.18704 |
| **BI** | Grade 10<br>Grade 11 | 23<br>22 | 2.6667<br>3.5909 | 1.01504<br>.94243 | .21165<br>.20093 |

As seen in Table 1, the mean values for constructs were calculated separately for Grades 10 and 11. The results show that the mean value for all questions describing the perceived usefulness of Scratch (U) was 1.98 and 2.34, respectively, for Grades 10 and 11. Both these values were below the midpoint (3), indicating a generally positive response. There was a statistically significant difference of perceived usefulness between the two grades (chi-square = 8.175, p = 0.004), with a mean rank of 17.56 for Grade 10 and 28.70 for Grade 11. Thus, Grade 10 students were more positive than Grade 11 students, to a statistically significant extent, towards the perceived usefulness of Scratch.

For perceived ease of use (E), the mean values for Grade 10 and 11 were 2.14 and 1.69 respectively. Both these values were below the midpoint (3), indicating a generally positive response. But there was no statistically significant difference of perceived ease of use between the two grades (chi-square = 3.679, p = 0.055), with a mean rank of 26.65 for Grade 10 and 19.18 for Grade 11.

The mean value for questions related to students' attitude towards using Scratch (A) was 2.2 for Grade 10, indicating a generally positive response. Meanwhile, the mean value for attitude for Grade 11 respondents was 2.71, indicating a generally positive (but not strongly positive) response. There was no statistically significant difference in attitude between the two grades (chi-square = 3.333, p = 0.068), with a mean rank of 19.52 for Grade 10 and 26.64 for Grade 11.

The mean value for questions related to students' behavioural intention to use Scratch (BI) was 2.66 for Grade 10, indicating a generally positive (but not strongly positive) response, while Grade 11 produced a mean value of 3.5, indicating a generally negative (but not strongly negative) response. There was a statistically significant difference in behavioural intention between the two grades (chi-square = 8.040, p = 0.005), with a mean rank of 17.61 for Grade 10 and 28.64 for Grade 11. These results show that the Grade 11 students, who had had experience with both Java and Scratch, had a more negative response, compared to the Grade 10 students, to perceived usefulness of Scratch and behavioural intention to use Scratch. Results from the open-ended questions were analysed to assess these phenomena.

Seventeen of the 22 Grade 11 students reported that they would have rather studied Java than Scratch in Grade 10. Respondent 5 from Grade 11 stated that "Java is a better programming language in general as it is more accurate and meticulous. Scratch is fairly simple and doesn't really contribute to learning programming skills well." Respondents 6 indicated that with Scratch "you didn't need to learn any coding". Respondents 7 said that "Scratch is more child-friendly". Respondent 10 described Scratch as "too junior", stating that it did not provide them with the sufficient level of programming knowledge they required for Java in Grade 11.

### *Correlation analysis*

A Spearman rank correlation coefficient was used to measure the strength and direction between the four TAM constructs. Scores from the Likert scale questions ranged from 1 (strongly agree) to 5 (strongly disagree). The overall mean scores for each construct were calculated and are presented in Table 2.

**Table 2: Correlation coefficients: Grade 10 respondents**

| Construct | Spearman correlation coefficients | Significance (2-tailed) |
|---|---|---|
| U – BI | 0.539** | 0.008 |
| E – BI | 0.291 | 0.178 |
| A – BI | 0.701** | 0.000 |

**Correlation is significant at the 0.01 level (2-tailed)

The results (Table 2) show there was a moderately positive association (0.539) between Grade 10 students' beliefs that they found Scratch to be useful (U) and their intention to use it (BI). This statistically significant (p=0.008) association indicates that perceived usefulness generally did influence students' behavioural intention to use the Scratch programming language.

There was a strongly positive association (0.701) between the Grade 10 students' attitude towards using Scratch (A) and their behavioural intention to use it (BI).

This statistically significant (p=0.000) association indicates that students' attitudes towards Scratch programming did significantly influence their behavioural intention to use it. However, the Grade 10 respondents' beliefs about Scratch's ease of use (E) did not correlate significantly (r=-0.058, p=0.703) with students' behavioural intention to use it (BI). This means that the there was almost no linear association between how easy the Grade 10 students found Scratch to use and their behavioural intention to use it, indicating that perceived ease of use did not appear to influence behavioural intention.

**Table 3: Correlation coefficients: Grade 11 respondents**

| Construct | Spearman correlation coefficients | Sigificance (2-tailed) |
|---|---|---|
| U – BI | -0.058 | 0.703 |
| E – BI | -0.225 | 0.315 |
| A – BI | 0.743** | 0.000 |

**Correlation is significant at the 0.01 level (2-tailed)

Table 3 presents the correlation coefficients for Grade 11 students. Unlike with Grade 10 students, perceived usefulness of Scratch (U) did not correlate significantly (r=-0.058, p=0.703) with Grade 11 students' behavioural intention to use it (BI). This means that the there was almost no linear association between how useful Grade 11 students found Scratch and their intention to use it, indicating that perceived usefulness did not appear to influence behavioural intention. This is presumably attributable to the Grade 11 students' experience with more advanced programming languages.

There was a strongly positive association (0.743) between Grade 11 respondents' attitude towards using Scratch (A) and their behavioural intention to use it (BI). This statistically significant (p=0.000) association indicates that Grade 11 students' attitude towards Scratch programming appeared to significantly influence their intention to use it.

Meanwhile, the Grade 11 students' beliefs about Scratch's ease of use (E) did not correlate significantly (r=-0.225, p=0.315) with their behavioural intention to use it (BI). This means that there was almost no linear association between how easy the Grade 11 respondents found Scratch to use and their intention to use it, indicating that perceived ease of use did not appear to influence Grade 11 students' behavioural intentions.

## 5. Conclusions and recommendations

This study aimed to identify how Scratch programming has been accepted by South African students, by exploring the associations between perceived usefulness, perceived ease of use, attitude towards using, and behavioural intention to use—the four factors specified by TAM. This study thus seeks to make a contribution to the literature on Scratch programming, and on its effects in South African schooling since its introduction in 2011. Findings from the study suggest that perceived usefulness and attitude towards using were both significant factors in influencing student respondents' (in both Grade 10 and 11) behavioural intention to use Scratch. The findings also showed that perceived ease of use did not significantly influence, among both Grade 10 and 11 respondents, behavioural intention to use Scratch.

When data from both the closed-ended (quantitative) and open-ended (qualitative) survey questions were considered together, it became clear that Grade 11 students showed a more negative response than Grade 10 students to Scratch's usefulness. Data derived from the survey questionnaire's open-ended questions provided indications that Grade 11 respondents' unfavourable view of Scratch's usefulness was attributable to the fact that, unlike Grade 10 students who had only had experience with Scratch, the Grade 11 students had also been exposed to Java and many had come to the belief that Scratch did not provide the required knowledge, both syntactic and computational, required for Java.

Thus, the overall findings suggest that while Scratch helps students understand logic and problem-solving, it does not, according to the Grade 11 respondents, assist sufficiently in preparing students for using a higher-level programming language such as Java. The findings showed that although Grade 11 students perceived Scratch to be useful, they found it did not prepare them for "real programming", with the majority of Grade 11 students stating that it would have been better to study Java, instead of Scratch, in Grade 10. The transition from Scratch to Java was found, therefore, to involve too large a gap, leading to students perhaps losing interest in programming and perhaps leading to them eventually changing to another subject. Koorsse et al. (2015) have found that due to South African students finding programming difficult, many students change to an easier subject in Grade 12, or remain attempting the subject while lacking motivation and interest.

Accordingly, our recommendation, based on the findings of this study, is that another programming language, such as Visual Basic or Delphi, be introduced after exposure to Scratch, before students move on to Java. This would serve as a sufficient intermediary platform, to bridge the complexity gap between Scratch and Java. This would also assist in the gradual development of programming skills, allowing for greater understanding that would increase satisfaction in these subjects for students. Another result would be increased readiness and throughput of students who want to pursue subjects in the computing discipline at tertiary level, hence assisting in addressing the skills shortage faced by the South African IT sector.

Since Scratch has been found to develop both logical and problem-solving skills in this and other studies (Calao et al., 2015; Kalelioğlu & Gülbahar, 2014; Lai & Yang, 2011; Calder, 2010), it would also be beneficial to introduce this programming language much earlier in the schooling careers of South African students than in Grade 10, thus potentially implicitly assisting students to acquire better understanding of subjects that require problem-solving and logical thinking.

It must be noted that this study had  limitations, due to its small sample size. The initial targeted sample size was 70, but it had to be reduced to 45 students because respondent students switched to another subject during the course of the year or did not appropriately complete the questionnaire. A more in-depth study could be carried out, including more students and more schools, thereby resulting in a larger sample size which would allow for better generalisability of findings.

In conclusion, this study provides insight into the perceptions of Scratch programming by students in two South Africa high schools, by highlighting factors that promote and inhibit its adoption by students.  The results of this study could be considered by South African education policymakers and curriculum developers, to help inform policies and curriculum aimed at the following goals: increasing the retention rate of students in programming subjects; and providing students with the necessary skills to succeed in their tertiary education, and in industry. Successful realisation of these goals can assist in building South African programming capacity and capabilities, addressing the shortage of programming skills  in the country, and decreasing the country's reliance on offshore-outsourcing of these skills.

## References

Annamalai, S., & Salam, S. N. A. (2017). Facilitating programming comprehension for novice learners with multimedia approach: A preliminary investigation. In American Institute of Physics (AIP) (Ed.), *AIP conference proceedings 1891*. https://doi.org/10.1063/1.5005362

Baytak, A., & Land, S. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development, 59*(7), 765–782. https://doi.org/10.1007/s11423-010-9184-z

Beyers, R. N., & Van der Merwe, L. (2017). Initiating a pipeline for the computer industry: Using Scratch and LEGO robotics. In IEEE (Ed.), *2017 Conference on Information Communications Technology and Society (ICTAS): Proceedings*. Umhlanga, Durban, 8–10 March. https://doi.org/10.1109/ictas.2017.7920646

Bosse, Y., & Gerosa, M. A. (2017). Why is programming so difficult to learn? Patterns of difficulties related to programming learning. *ACM SIGSOFT Software Engineering Notes, 41*(6), 1–6. https://doi.org/10.1145/3011286.3011301

Boyle, T., Bradley, C., Chalk, P., Jones, R., & Pickard, P. (2003). Using blended learning to improve student success rates in learning to program. *Journal of Educational Media (Special Edition on Blended Learning), 28*(2), 165–178. https://doi.org/10.1080/1358165032000153160

Cakir, R., & Solak, E. (2014). Exploring the factors influencing e-learning of Turkish EFL learners through TAM. *The Turkish Online Journal of Educational Technology*, *13*(3), 79–87. https://doi.org/10.1016/j.sbspro.2015.01.515

Calao, L. A., Moreno-León ,J., Correa, H. E., & Robles, G. (2015). Developing mathematical thinking with Scratch: An experiment with 6th Grade students. In G. Conole, T. Klobučar, C. Rensing, J. Konert, & E. Lavoué (Eds.), *Proceedings of 10th European Conference on Technology Enhanced Learning (EC-TEL 2015)*, Toledo, Spain, September 15-18 (pp. 17–25).

Calder, N. (2010). Using Scratch: An integrated problem-solving approach to mathematical thinking. *Australian Primary Mathematics Classroom, 15*(4), 9–14.

Chetty, J., & Barlow-Jones., G. (2012). Integrating teaching-and-learning techniques for novice computer programming students. In Advancement of Computing in Education (AACE) (Ed.), *Proceedings of Global Learn 2012: Global Conference on Learning and Technology* (pp. 142-146). Retrieved from https://www.learntechlib.org/primary/p/42056

Chok, N. S. (2010). Pearson's versus Spearman's and Kendall's coefficients for continuous data. Retrieved from http://d-scholarship.pitt.edu/8056/1/Chokns_etd2010.pdf

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly, 13*(3), 319–340. https://doi.org/10.2307/249008

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: A comparison of two theoretical models. *Management Science*, *35*(8), 982–1003. https://doi.org/10.1287/mnsc.35.8.982

Department of Basic Education. (2011). Curriculum and Assessment Policy Statement (CAPS). Pretoria: Government of South Africa.

Ebrahimi, A., Geranzeli, S., & Shokouhi, T. (2013). Programming for children: "Alice and Scratch analysis". Paper presented to the 3rd International Conference on Emerging Trends of Computer and Information Technology (ICETCIT), Singapore, 6–7 November.

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

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87–97. https://doi.org/10.1016/j.compedu.2012.11.016

Housand, B. C., & Housand, A. M. (2011). Plugging into creative outlets. *Gifted Education Communicator*, *42*(1), 20–23.

Ibrahim, R., Leng, N. S., Yusoff, R. C. M., Samy, G. N., Masrom, S., & Rizman, Z. I. (2017). E-learning acceptance based on technology acceptance model (TAM). *Journal of Fundamental and Applied Sciences, 9*(4S), 871–889.

Isa, N. A. M., & Derus, S. R. M. (2017). Students experience in learning fundamental programming: An analysis by gender perception. *Advanced Journal of Technical and Vocational Education, 1*(1), 240–248.

Iskrenovic-Momcilovic, O. (2017). Choice of visual programming language for learning programming. *International Journal of Computers, 2*, 250–254.

Kalelioglu, F., & Gulbahar, Y. (2014). The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13*(1), 33–50.

Kim, H., Choi, H., Han, J., & So, H. (2011). Enhancing teachers' ICT capacity for the 21st century learning environment: Three cases of teacher education in Korea. *Australasian Journal of Educational Technology (AJET), 28*, 965–982. https://doi.org/10.14742/ajet.805

Koorsse, M., Cilliers, C., & Calitz, A. (2015). Programming assistance tools to support the learning of IT programming in South African secondary schools. *Computers and Education, 82*, 162–178. https://doi.org/10.1016/j.compedu.2014.11.020

Lai, A.-F., & Yang, S.-M. (2011). The learning effect of visualized programming learning on 6th graders' problem solving and logical reasoning abilities. In IEEE (Ed.), *2011 International Conference on Electrical and Control Engineering (ICECE),* Yichang, China, 16-18 September (pp. 6940–6944). https://doi.org/10.1109/iceceng.2011.6056908

Lamb, A., & Johnson, L. (2011). Scratch: Computer programming for 21st century learners. *Teacher Librarian, 38*(4), 64–68.

Lee, Y. J. (2011). Scratch: Multimedia programming environment for young gifted learners. *Gifted Child Today, 34*(2), 26–31. https://doi.org/10.1177/107621751103400208

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. *SIGCSE'10* Retrieved from http://ims.mii.lt/ims/konferenciju_medziaga/SIGCSE'10/docs/p346.pdf

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

Korkmaz, Ö. (2016). The effect of Scratch- and Lego Mindstorms Ev3-based programming activities on academic achievement, problem-solving skills and logical-mathematical thinking skills of students. *MOJES:Malaysian Online Journal of Educational Sciences, 4*(3), 73–88.

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. Paper presented to SIGCSE'07, 7-10 March, Covington, KY. Retrieved from https://cs.harvard.edu/malan/publications/fp079-malan.pdf

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

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education, 16*(3), 211–227. https://doi.org/10.1080/08993400600912384

Marques, O. F., & Marques, M. T. (2012). No problem? No research, little learning ... big problem! *Systemics, Cybernetics and Informatics, 10*(3), 60–62.

McCracken, M., Almstrum, V., Daiz, D., Guzdail, M., Hagan, D., Kolikant, Y. B., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. Paper presented at the Working group reports from ITiCSE on Innovation and technology in computer science education, Canterbury, UK. https://doi.org/10.1145/572134.572137

Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education, 23*(3), 239–264. https://doi.org/10.1080/08993408.2013.832022

Meyerovich, L., & Rabkin, A. (2013). Empirical analysis of programming language adoption. Retrieved from http://sns.cs.princeton.edu/docs/asr-oopsla13.pdf

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—Views of students and tutors. *Education and Information Technologies, 7*(1), 55–66.

Moser, R. (1997). A fantasy adventure game as a learning environment: Why learning to program is so difficult and what can be done about it. In Special Interest Group on Computer Science Education (SIGCSE) (Ed.), *ITiCSE '97:Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education* (pp. 114-116), Uppsala, Sweden, 1–5 June. https://doi.org/10.1145/268809.268853

Mugo, D. G., Njagi, K., Chemwei, B., & Motanya, J. O. (2017). The technology acceptance model (TAM) and its application to the utilization of mobile learning technologies. *British Journal of Mathematics & Computer Science, 20*(4), 1–8. https://doi.org/10.9734/bjmcs/2017/29015

Olivier, J. (2016). Blended learning in a first-year language class: Evaluating the acceptance of an interactive learning environment. *Journal of Literary Criticism, Comparative Linguistics and Literary Studies, 32*(2), 1–12. https://doi.org/10.4102/lit.v37i2.1288

Ozmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: Difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry, 5*(3), 9–27.

Papadopoulos, Y., & Tegos, S. (2012). Using microworlds to introduce programming to novices. In IEEE (Ed.), *Proceedings of the 2012 16th Panhellenic Conference on Informatics,* Piraeus, Greece, 5–7 October. https://doi.org/10.1109/pci.2012.18

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. New York: Basic Books.

Parsons, D., & Haden, P. (2007). Programming osmosis: Knowledge transfer from imperative to visual programming environments. In S. Mann, & N. Bridgeman (Eds.), *Proceedings of the Twentieth Annual NACCQ Conference,* Hamilton, New Zealand (pp. 209–215).

Pendergast, M. O. (2006). Teaching introductory programming to IS students: Java problems and pitfalls *Journal of Information Technology Education, 5*(1), 491–515. https://doi.org/10.28945/261

Permatasari, L., Yuana, R. A., & Maryono, D. (2018). Implementation of Scratch application to improve learning outcomes and student motivation on basic programming subjects. *Indonesian Journal of Informatics Education, 2*(2), 95–102.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Rudder, A., Bernard, M., & Mohammed, S. (2007). Teaching programming using visualization. In ACTA (Ed.), *Proceedings of the Sixth Conference on IASTED International Conference Web-Based Education - Volume 2* (pp. 487–492), Chamonix, France, 14-16 March.

Sáez-López, J. M., 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. https://doi.org/10.1016/j.compedu.2016.03.003

Sanchez-Mena, A., Marti-Parreno, J., & Aldas-Manzano, J. (2017). The effect of age on teachers' intention to use educational video games: A TAM approach. *The Electronic Journal of e-Learning, 15*(4), 355–366.

Tan, P.-H., Ting, C.-Y., & Ling, S.-W. (2009). Learning difficulties in programming courses: Undergraduates' perspective and perception. In IEEE (Ed.), *ICCTD 2009: 2009 International Conference on Computer Technology and Development, Volume 2* (pp. 42–46). https://doi.org/10.1109/icctd.2009.188

Tavakol, M., & R. Dennick, R. (2011). Making sense of Cronbach's alpha. *International Journal of Medical Education, 2*, 53–55. https://doi.org/10.5116/ijme.4dfb.8dfd

Theodorou, C., & Kordaki, M. (2010). Super Mario: A collaborative game for the learning of variables in programming. *International Journal of Academic Research, 2*(4), 111–118.

Tom, M. (2015). Five Cs Framework: A student-centered approach for teaching programming courses to students with diverse disciplinary background. *Journal of Learning Design, 8*(1), 21–37. https://doi.org/10.5204/jld.v8i1.193

Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education, 120*, 64–74. https://doi.org/10.1016/j.compedu.2018.01.011

Van Zyl, S., Mentz, E., & Havenga, M. (2016). Lessons learned from teaching Scratch as an introduction to object-oriented programming in Delphi. *African Journal of Research in Mathematics, Science and Technology Education, 20*(2), 131–141. https://doi.org/10.1080/18117295.2016.1189215

Watters, A. (2011). Scratch: Teaching the difference between creating and remixing. Mind/Shift. Retrieved from http://ww2.kqed.org/mindshift/2011/08/11/scratch-teaching-kids-about-programming-teaching-kids-about-remixing/

Wilson, A., & Moffat, D. C. (2010). *Evaluating Scratch to introduce younger schoolchildren to programming*. School of Engineering and Computing, Glasgow Caledonian University, Glasgow. Retrieved from http://scratched.gse.harvard.edu/sites/default/files/wilson-moffat-ppig2010-final.pdf