Scientific Research Publishing

Open Journal of Social Sciences, 2022, 10, 476-486
https://www.scirp.org/journal/jss
ISSN Online: 2327-5960
ISSN Print: 2327-5952

Unplugged Teaching: Deepening Information Technology Learning

Reuben Dlamini, Alton Dewa

University of the Witwatersrand, Johannesburg, South Africa
Email: Reuben.Dlamini@wits.ac.za, alton.dewa@wits.ac.za

How to cite this paper: Dlamini, R., & Dewa, A. (2022). Unplugged Teaching: Deepening Information Technology Learning. *Open Journal of Social Sciences, 10,* 476-486.
https://doi.org/10.4236/jss.2022.104034

Received: March 17, 2022
Accepted: April 26, 2022
Published: April 29, 2022

Abstract

There has been a decline in the number of public schools in South Africa offering information technology as a subject, yet it holds a special significance in higher level computational courses especially at tertiary level. Learners doing programming in information technology (IT) have difficulties during the initial stages of learning how to design and develop programs due to the abstract nature of the subject and the demands on the use of data structures to establish needed heuristics to solve problems or design solutions. Through the Semantic wave code of Legitimation Code Theory lens, this research aims to address the cognitive aspects of deeper information technology learning especially acquiring the appropriate knowledge structures for program design and problem-solving. The abstract semantics are the cornerstone of information technology as a school subject and their difficulty may cause memory overload, hence the importance of mental structures on how computers work. Through the semantic wave construct of the Legitimation Code Theory, we unpacked unfamiliar settings and repacked the different settings by moving from simpler meanings [concrete] to complex meanings [abstract]. In this work, we discuss issues that make programming more accessible by navigating the abstract semantics using contextualised exploration giving learners' concretized understanding while placing them in unfamiliar settings requiring creativity. Importantly, this work builds on researched literature and experience as information technology subject specialists explaining how we subject learners through various context and semantic profiles due to the multi-level nature of abstraction in programming.

Keywords

Abstraction, Algorithmic Design, Programming, Flowchart, Legitimate Code Theory, Semantic Wave

1. Introduction

Computer science is increasingly being taught in high schools around the world, yet there is little research on effective teaching, and many teachers lack relevant experience (Waite, Maton, Curzon, & Tuttiett, 2019). Different teaching approaches have emerged, with various degrees of success. During teaching and learning, programming ideas in computer science constitute a challenge to both educators and learners. Simple teaching approaches are necessary to aid teachers in determining the efficacy of lesson plans and identifying strategies to enhance them. Programming languages help students develop procedural knowledge needed to establish good problem-solving skills gradually from one level to the next. Many students shun studying computer science (CS) or information technology (IT) at high school because of the presence of complexity and difficulty associated with programming languages. In most cases, students become helpless, unable to deal with a new language with unfamiliar syntax.

Even, contemporary approaches such as student-centred are not yielding the desired results for disciplinary knowledge. In the effort to make the information technology accessible to all learners, we treat the subject as a "construction of hierarchically ordered collections of specific skills" (Fischer, 1980: p. 477). Fundamentally, to become an information technology, teacher must have access to opportunities at various contexts in the sequence of information technology curriculum from simpler meanings [concrete] to complex meanings [abstract]. Therefore, it is important to understand how people learn and the nature of knowledge to be presented. It is generally accepted to have deeper understanding of the complexity of concepts and elements of information technology; hence, in this work, we adopted the Semantic code of the Legitimation Code Theory to reduce complex cognitive tasks to more manageable load levels.

The Semantic Code of the Legitimation Code Theory (LCT) provides fascinating insight into how to make teaching programming languages exciting for students, as well as the ways in which information may be combined to create meaning. LCT allows systematized thoughts to be diagrammatically depicted, allowing for better communication. LCT may provide a conceptual framework for delving underneath empirical manifestations to reveal epistemic and semantic principles that can improve students' enjoyment of programming languages. In other words, LCT is a framework for interrogating what establishes and confirm good learning experience (Maton, 2013). In this paper, we begin by explaining what computer-programming languages are (not limited to a specific programming language) and how they work. Various problems experienced by the learners are highlighted including concepts such as program design and algorithms. In addition, we engage the literature to bring out why learners find the computer programming complex and challenging. Last but not least, we provide an unplugged pedagogical approach to teaching programming concepts, in which students are given real, tangible exercises or demonstrations to convey abstract, intangible computing concepts while also allowing them to explore powerful

ideas. This method is becoming widely used in the world, while evidence of its efficacy is equivocal.

2. Computer Programming Languages

A computer is a general-purpose machine capable of doing a wide range of computations. In a fraction of a second, contemporary computers can do billions of computations. These machines are incapable of doing anything on their own. In San Ahmed et al. (2018), Renumol, Jayaprakash, and Janakiram (2009) described computer programming as "the act of writing, testing, and debugging computer programs utilizing various programming languages" (p. 27). An instruction must be supplied to a computer in order for it to do a task. These instructions contain step by step information to perform a specific task. Computers and smart phones has some sort of code behind it telling it what to do (Malik, 2011). There are thousands of languages in the world around us and so is in digital technologies. There are numerous types of programming languages making up the code as powerful as that technology. A programming language is made up of specific terms and directions used to create some kind of output such as websites, applications (apps), or any kind of software. A programming language, in general, is an artificial language used to send instructions to a machine. They can also be used to make programs that regulate the machine's behavior (for example, computer, robot, TV rocket, etc.). A computer program is a set of instructions expressed in a programming language that is used to control a machine's behavior. There are many different types of programming languages such as Java, Python, C/C++, PhP, C#, Pearl to name a few. All of these languages are unique and operate differently from one another. For example, Web developer may choose to use Java Script because it works well with HTLM or CSS. A video game designer may choose C++ because it can handle more graphics that are complex. Data Analysts, Engineers, Scientist, and Mathematicians may choose Python, R or Matlab. Low level and high level programming languages are the two categories of programming languages. Low-level language, often known as assembly or machine language, is the language that machines understand. High-level languages are easier to use and are more similar to human language.

3. Challenges in Writing Programming Codes

Programming is a staple skill very useful for recent years and its demand is high as new technologies are developed (Förster, Förster, & Löwe, 2020). However, programming courses are "generally regarded as difficult, and often have the highest dropout rates" (Robins, Rountree, & Rountree, 2003: p. 137). In addition, Papadakis & Orfanakis (2018) reiterate that "computer programing is difficult to learn and programing courses often have high drop-out rates" (p. 1). This assertion is supported by Costa, Aparicio, & Cordeiro (2012) who also found that even educators "face problems when teaching students programming mainly because programming is dynamic and abstract" (p. 25). Costa et al.

(2012) mention that "it is quite common that students start studying programming by features and variables rather than modelling" (p. 25). Despite the fact that learners may demonstrate understanding of the programming language syntax, they lack understanding of combing syntax and semantics of individual statements to produce running programs (Soloway & Spohrer, 2013). Programming requires a good analytical skill which is backed up by cognitive domain. There is a need to come up with variables that will store various data of the program. Some complexes of the programming language include interpretation of the requirements of the problem, translating the problem statement to logic program, designing algorithm and analyzing the program structure. The most difficulties in programming is the result of lack of skills largely be educators. They have certain difficulties in merging program design and comprehension knowledge (Costa et al., 2012). Some educators struggle with deciding what kinds of materials to use to encourage students to enjoy learning computer programming languages and, as a result, improve learning results (San Ahmed, Sardasht, Mahmood, Nabi, & Hussein, 2018).

Programming is considered the complex and difficult subject due to its rigidity and also needs multiple abilities and knowledge from other subject disciplines like math, science, etc. Both educators and learners need to have declarative and procedural knowledge (San Ahmed et al., 2018). Declarative knowledge refers to knowing the syntax of the programming language and comprehend semantics. While procedural knowledge is about "abstraction and logical thinking skills" that help designing and solving program (Norvig, 2001). Learners lack logical reasoning and abstraction due to limited mathematical problem-solving competences, which are considered essentials for any programming language (Gomes & Mendes, 2007). In addition, new students in programming have no idea of how programming structure works, or to implement syntax structures correctly and detect and debug programming errors. Educators' teaching approach can pose challenges when teaching programming. Majority of educators still use traditional method of teaching that uses PowerPoint as a mode of teaching. Learners of the 21st century prefer learning that is interactive, making them to be constructors of their own knowledge not to be passive recipient of knowledge from the teachers. Such teaching approach is considered "boring and tedious". Yet the role of the educators is to build and transfer the right knowledge to students prior having understanding of programming concepts such as the ability to write a code that can solve the problem.

4. Teaching and Learning Programming

The resurgence of computer programming at higher institutions has prompted schools to include it in their curricula, promising to prepare students for a "future that goes beyond learning how to code" (Popat & Starkey, 2019). Knowing how to write programming code prepares the future workforce to supply the IT industry and IT-compliant workers of the future (Balanskat & Engelhardt, 2014).

New roles for the people are required to bring the newly introduced technol-

ogy to life and maintenance. Learning the syntax of any programming language is one of the needed skills. Different programming languages have different syntax or syntactical paradigm. Other important skills include knowing algorithms and data structures of any programming languages. This technique helps learners to convert the problem statement into a solution. Once algorithms have been designed, they can be converted into a code as the teacher teaches the coding live, learners can identify errors in the process, and may work together in debugging the code, correcting the errors in the code so that the program can run successfully (Raj et al., 2018). Thus, this kind of practice exposes students to good programming skills. Schröer & Koschke (2021) concur with Raj et al. (2018) that it is necessary for learners to observe what experts in programming do during programming processes, i.e., they need to see what specific actions and strategies they apply to solve a task at hand.

The other method to right path of learning the programming language is to start with basic programming fundamental such as writing a code that output "Hello, World", adding and subtracting numbers. Once these concepts are mastered, they give learners confidence to move to the next level of challenging concepts. If the basics are not strong, there are likely to impede the progress of the learner when dealing with advanced concepts. The student has to give himself/herself a minimum of 5 to 10 days learning fundament basics of programming. Practising writing programming language is essential. The information practiced tends to be in the memory for a long period, and this is better than memorising the syntax. From there can start attempting challenging concepts and start doing a simple project where they synthesise learned concept to build a software.

Programming is practical and is learnt by doing. The only thing that matters is that the place where one starts fitting into current level of knowledge. In other words, a programmer who really knows a person who really knows five of six languages can learnt the seventh without having to go through all the basics. Programming is an incredible skill needed in this 21st century. Programming provides an opportunity to build something right (for example, an app or a game) and express oneself clearly. However, during the process of writing codes, mistakes occur, thus another critical area those teaching programming should take cognisance of. They learn from the mistakes and errors that can affect the running of the program. Generally, to program allows transmitting the idea that is in the head and turn it into series of steps that the computer can understand and execute. Programming languages have different syntax and different rules, however, they achieve one purpose, to communicate something to the computer to do something.

5. Legitimate Code Theory

Majority of learners in South Africa face challenges of understanding programming as they transition from senior phase (Grades 7 - 9) to Further Education and Training (FET) phase (Grades 10 - 12). The probability of successful transition is dependent on learners' prior knowledge about the subject and how this

knowledge can be used to learn a new subject (Mouton & Archer, 2019). Programming in South Africa formally starts at 10. At this grade, learners are clueless or have absolutely no knowledge what programming is and its expectations. This paper brings deepening of teaching of the programming languages by drawing on Legitimation Code Theory (LCT) as lens for changing of pedagogy. Many learners especially from disadvantaged educational backgrounds find programming a challenge resulting in massive dropout rates of the Information Technology (IT).

LCT is a sociological framework that promises a reliable method of study, as well as the ability to adapt and shape teaching practice, among other things (Maton, 2013, 2014b). It is based on a variety of ideas, including Bourdieu's field theory and Bernstein's coding theory (Maton, 2013, 2014b). Knowledge is viewed as a socially created object in LCT (Mouton & Archer, 2019). Different dimensions can be utilized to examine certain sets of organizing principles, as well as underlying practices that serve as legitimation codes (Maton, 2014b). LCT is a conceptual framework that affords interrogation of knowledge practices. It critiques what makes knowledge legitimate. LCT gives a frame of beginning to understand problems the concept through use of graph representations, which can be used in any educational enterprise. The code part in LCT comes with bunch of different codes, for example, specialisation code and semantic code can apply in teaching programming languages. Specialisation code essentially means the dynamic epistemic relationships and social relationships. Specialisation codes is about what make someone or something special, distinct and worth of status (Maton, 2013). Whoever controls the device establishes what knowledge is legitimate. It begins from a simple idea that every knowledge claim or practice is by someone and about or oriented towards something. Therefore, any knowledge claim or practice sets up relation to an object and to a subject

These relations are called epistemic relations between knowledge claims and that part of the world they are about. In the case of knowledge claims, they manifest as epistemic linkages between knowledge and its declared object of study, as well as social relations between knowledge and its authors or subjects (Maton & Chen, 2017). The practice emphasizes each of these relations in different ways, that is, each may be more strongly or weakly emphasized. Thus, epistemic relations are about specialist knowledge such as specific skills, technique and procedures. What is being said or done is effectively known as legitimate. Social relations are about attributes of knowers (Maton, 2013). It is something about who is speaking, who is doing something and what is he/she doing. Thus, legitimacy depends on personal experience, natural ability or being a particular kind of person such as gender.

The degree of intricacy of activities, whether symbols, concepts, or sentences, is referred to as semantic density. It defines how many meanings are packed into or condensed into something (**Figure 1**). As a result, the higher the semantic density (SG+), the more meanings are compressed. The lower the semantic

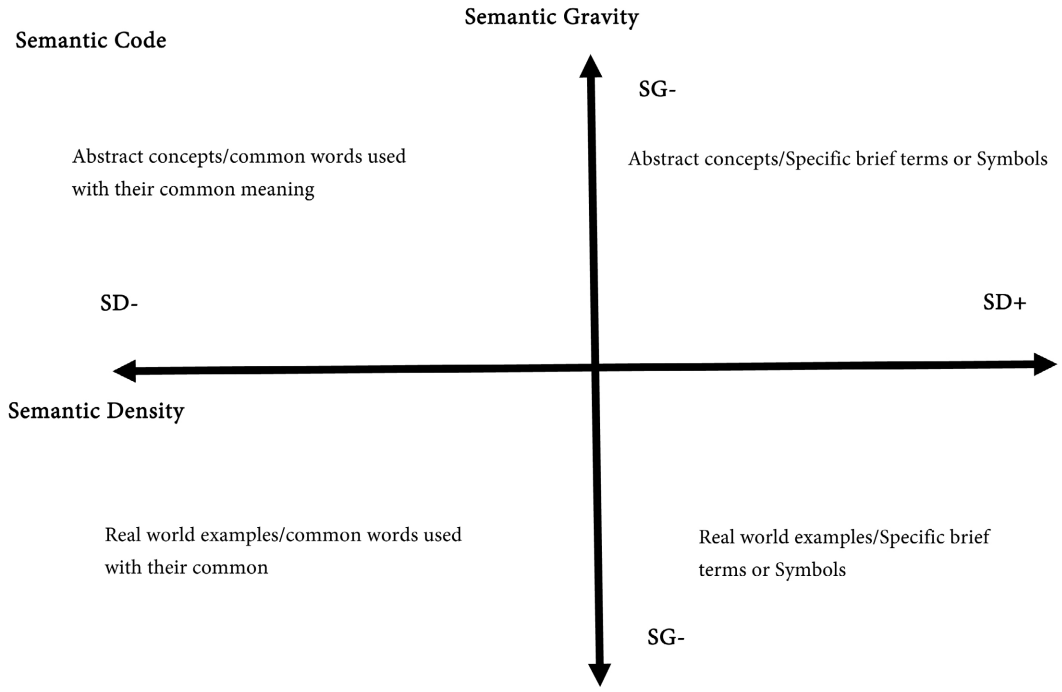

Figure 1. Semantic code quadrant: Source: Maton, Hood & Shay (2016).

gravity (SG−), the less meaning is contained in it (Maton, 2013, 2014a; Maton, Hood, & Shay, 2016).

Semantic Wave

Semantic wave theory is a simple yet effective method for teaching concepts in the context of learning, with an emphasis on unplugged programming activities (Maton, Hood, & Shay, 2016). The semantic wave is interested in what makes a good explanation or learning experience, whether textual, multimedia, or an online lecture. It also helps to figure out why unplugged education works in some circumstances and doesn't in others. Students can utilize semantic waves to evaluate lesson plans, online resources, and to learn how to create clear and understandable explanations. Teachers, for example, can use learning activities or the plan for learning activities and follow it to accomplish the goals. The abstract principles are gradually presented through real examples and demonstrations (Martin, Maton, & Doran, 2019). It is comparable to taking learners on a journey where they begin as novices, unsure about programming principles, but at the end of the journey, they will have a better understanding of programming concepts (see **Figure 2**).

Learning a subject entails understanding both the language and concepts, as well as the abilities that go with them. Learners must be supplied with technical notions in order to go from novice to master of programming subject. It is suggested by semantic wave theory that explanations begin at an abstract level and progress to a concrete level or daily language. With time, the abstract ideas or what learners may understand are linked back to the familiar notions or language.

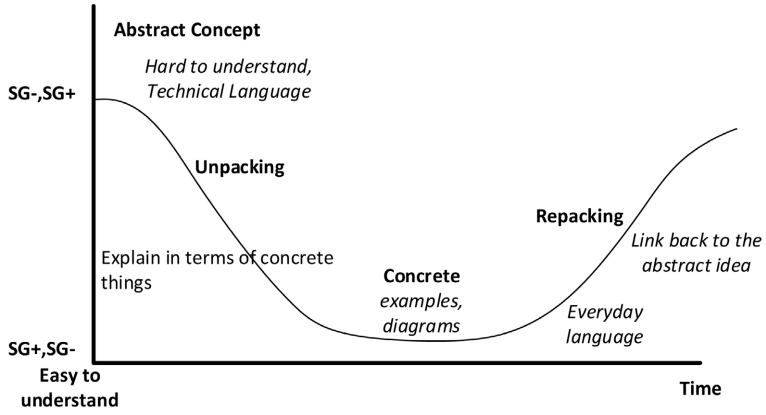

Figure 2. Semantic wave: Source: Waite, Maton, Curzon, & Tuttiett (2019).

The objective is to unpack the concept and make the concealed concept evident in learners' minds (Martin, Maton, & Doran, 2019). At this level, however, learning occurs only when it is repackaged, that is, when it is related to abstract concepts (Waite, Maton, Curzon, & Tuttiett, 2019). Teaching, according to this view, begins with the introduction of abstract concepts. According to this idea, education begins with the introduction of abstract concepts in technical language that learners must understand, and is then presented in concrete (connecting it to abstract) concepts through the use of explanation, demonstration, modelling, and other methods. The learners are then given the opportunity to make connections between the concrete and abstract concepts (Maton, Hood, & Shay, 2016). Semantic wave provides a quick and simple but powerful way of evaluating learning activities. It gives an insight to why certain concepts worked or not worked during teaching and learning. In other words, it supports reflective changes to improve teacher activities. Before the final repacking, learners are encouraged to do repacking during the role play, which will help them get a deeper understanding of the sub-concepts in the context of the role play activities.

Good learning activities are not just the single wave; there are actually lots of waves between waves, linking onto the next wave. An instance of using a down escalator method leaves learners down at the bottom and never re-pack the concepts to abstract ideas is not ideal when teaching programming. Neither does the flat lining at the top nor bottom is ideal in teaching programming. Flat lining at the top emphasizes technical language or abstract ideas that will pose a challenge to the learners to grasp. Bottom lining concentrates on the use of concrete objects without linking them to abstract ideas. Hence, such teaching will not be complete, as it does not connects to other concepts. Teaching programming should be accompanied by metaphors, unplugged activities, and convert them into tangible concepts that learners can see (Martin, Maton, & Doran, 2019). For example, when teaching the concepts of variables in programming, explain the use of letters and what type of data type they will store. Assign data to declared variables and display the content of the variables. Learners can then be given

their own activities to practice the learned concept. By doing this they become the mastery of the concepts. Teachers are encouraged to start by outlining the learning outcomes, for instance variables in programming. The explanation of the variables and how they store information increases the abstraction. There is a need to explain this concept using words learners could understand without jargon hindrance. In this case, learners could see what is happening in each line of the code (Curzon, McOwan, Donohue, Wright, & Mars, 2018).

When teaching, it is important to move between these semantic codes quadrants. Moving explicitly forward and backward between the quadrants may help learners understand the concept taught. Use of simple language helps significantly as learners can later explain the concepts on their own. Teaching using semantic gravity and semantic density and moving between weak semantic gravity and high semantic gravity and low semantic density and high semantic density make learners incorporate the knowledge (Curzon, McOwan, Donohue, Wright, & Mars, 2018). The reaching is not unidirectional, meaning from real world to abstract, but can also move bidirectional, abstract to real world and vice-versa.

6. Summary

This work explains how we subject learners through various context and semantic profiles due to the multi-level nature of abstraction in programming. Fundamentally, to become an information technology teacher must have access to opportunities at various contexts in the sequence of information technology curriculum from simpler meanings to complex meanings. The semantic wave allows information technology educators to traverse the process of developing programming concepts and elements using more and fewer details as necessary. Thus, it is important of packing, unpacking and repacking oscillating between different programming principles and concepts. To achieve meaningful learning of programming as the key component of the information technology as a subject fits into the Semantic code, however teachers must be subjected to a rigorous training and development. The rigorous training and development will allow participants to develop deeper understanding and the knowledge structures of programming concepts and information technology subject elements. In our engagement with the literature, there is evidence that learning programming is a complex cognitive process.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

Balanskat, A., & Engelhardt, K. (2014). Computing Our Future: Computer Programming and Coding-Priorities, School Curricula and Initiatives across Europe. European Schoolnet.

Costa, C. J., Aparicio, M., & Cordeiro, C. . (2012, June). A Solution to Support Student Learning of Programming. In *Proceedings of the Workshop on Open Source and Design of Communication* (pp. 25-29). https://doi.org/10.1145/2316936.2316942

Curzon, P., McOwan, P. W., Donohue, J., Wright, S., & Mars, D. W. (2018). Teaching Computer Science Concepts. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer Science Education: Perspectives on Teaching and Learning in School.* London: Bloomsbury Publishing.

Fischer, K. W. (1980). A Theory of Cognitive Development: The Control and Construction of Hierarchies of Skills. *Psychological Review, 87,* 477. https://doi.org/10.1037/0033-295X.87.6.477

Förster, E. C., Förster, K. T., & Löwe, T. (2020). Teaching Programming Skills in Primary School Mathematics Classes: An Evaluation Using Game Programming. In *2018 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1504-1513). IEEE. https://doi.org/10.1109/EDUCON.2018.8363411

Gomes, A., & Mendes, A. J. (2007, September). Learning to Program-Difficulties and Solutions. In *International Conference on Engineering Education* (Vol. 7). CEE.

Malik, D. S. (2011). *JavaTM Programming: From Problem Analysis to Program Design*. Cengage Learning.

Martin, J. R., Maton, K., & Doran, Y. J. (2019). *Accessing Academic Discourse: Systemic Functional Linguistics and Legitimation Code Theory*. Routledge. https://doi.org/10.4324/9780429280726

Maton, K. (2013). Making Semantic Waves: A Key to Cumulative Knowledge Building. *Linguistics and Education, 24,* 8-22. https://doi.org/10.1016/j.linged.2012.11.005

Maton, K. (2014). *Knowledge and Knowers: Towards a Realist Sociology of Education*. Oxon: Routledge. https://doi.org/10.4324/9780203885734

Maton, K. (2014b). A Tall Order? Legitimation Code Theory for Academic Language and Learning. *Journal of Academic,* 34-48.

Maton, K., & Chen, R. T. H. (2017). *Specialization from Legitimation Code Theory: How the Basis of Achievement Shapes Student Success*. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=Specialization+codes%3A+Knowledge%2C+knowers+and+student+success&btnG=

Maton, K., Hood, S., & Shay, S. (2016). *Knowledge-Building*. https://doi.org/10.4324/9781315672342

Mouton, M., & Archer, E. (2019). Legitimation Code Theory to Facilitate Transition from High School to First-Year Biology. *Journal of Biological Education, 53,* 1-20. https://doi.org/10.1080/00219266.2017.1420681

Norvig, P. (2001). *Teach Yourself Programming in Ten Years*. http://norvig.com/21-days.html#answers

Papadakis, S., & Orfanakis, V. (2018). Comparing Novice Programing Environments for Use in Secondary Education: App Inventor for Android vs. Alice. *International Journal of Technology Enhanced Learning, 10,* 44-72. https://doi.org/10.1504/IJTEL.2018.088333

Popat, S., & Starkey, L. (2019). Learning to Code or Coding to Learn? A Systematic Review. *Computers & Education, 128,* 365-376. https://doi.org/10.1016/j.compedu.2018.10.005

Raj, A. G. S., Patel, J. M., Halverson, R., & Halverson, E. R. (2018, November). Role of Live-Coding in Learning Introductory Programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (pp. 1-8).

https://doi.org/10.1145/3279720.3279725

Renumol, V., Jayaprakash, S., & Janakiram, D. (2009). *Classification of Cognitive Difficulties of Students to Learn Computer Programming*. India: Indian Institute of Technology.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13,* 137-172. https://doi.org/10.1076/csed.13.2.137.14200

San Ahmed, R. A. M., Sardasht, M., Mahmood, R., Nabi, R. M., & Hussein, D. L. (2018). The Impact of Teaching Materials on Learning Computer Programming Languages in Kurdistan Region Universities and Institutes. *Kurdistan Journal of Applied Research, 3,* 1-7. https://doi.org/10.24017/science.2018.2.3.1

Schröer, M., & Koschke, R. (2021, March). Recording, Visualising and Understanding Developer Programming Behaviour. In 2*021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 561-566). IEEE. https://doi.org/10.1109/SANER50967.2021.00066

Soloway, E., & Spohrer, J. C. (2013). *Studying the Novice Programmer*. Psychology Press. https://doi.org/10.4324/9781315808321

Waite, J., Maton, K., Curzon, P., & Tuttiett, L. (2019, September). Unplugged Computing and Semantic Waves: Analyzing Crazy Characters. In *Proceedings of the 1st UK & Ireland Computing Education Research Conference* (pp. 1-7). https://doi.org/10.1145/3351287.3351291