Open Access
Baghdad Science Journal **Vol.16 (Special Issue) 2019**

DOI: http://dx.doi.org/10.21123/bsj.2019.16.2(SI).0453

Social Worked-Examples Technique to Enhance Student Engagement in Program Visualization

Abdullah Al-Sakkaf* ***Mazni Omar**** ***Mazida Ahmad*****

Received 14/9/2018, Accepted 7/11/2018, Published 20/6/2019

Abstract:

Learning programming is among the top challenges in computer science education. A part of that, program visualization (PV) is used as a tool to overcome the high failure and drop-out rates in an introductory programming course. Nevertheless, there are rising concerns about the effectiveness of the existing PV tools following the mixed results derived from various studies. Student engagement is also considered a vital factor in building a successful PV, while it is also an important part of the learning process in general. Several techniques have been introduced to enhance PV engagement; however, student engagement with PV is still challenging. This paper employed three theories—constructivism, social constructivism and cognitive load to propose a technique for enhancing student engagement with program visualisation. The social worked-examples (SWE) technique transforms the traditional worked-example into a social activity, whereby a greater focus is placed on the collaboration role in constructing students' knowledge. This study identified three principles that could enhance student engagement through the SWE technique: active learning, social collaboration and low-load activity.

Key words: Engagement, Introductory Programming Education ,Program Visualization, Worked-example

Introduction:

Learning programming is among the top challenges in computer science education (CSE). Researchers have reported that the difficulty in learning and teaching programming is due to several factors categorised under the nature of programming, problems relating to the student, and the teaching method implemented (1). Programming incorporates several complex skills, such as planning, program design and problem-solving; hence, novice programmers often find programming difficult at first. Furthermore, it is acknowledged as a complex cognitive task that requires a knowledge of how programs are executed. Further, novices tend to be limited with regards to the knowledge of a programming language. In addition, they often lack the mental models necessary to problem solve and find it difficult to grasp programming concepts.

Universiti Utara Malaysia, Malaysia
Corresponding author:* abbdullah1@hotmail.com
** mazni@uum.edu.my
*** mazida@uum.edu.my

Education technology is increasingly becoming a vital solution used to improve learning skills the overheads in terms of learning the new programming language syntax and semantics, as well as learning the programming tools and environment and developing problem-solving skills make learning programming an exhausting task for novices. As a result, this can lead to anxiety, frustration, fear, and demotivation (2,3).

Among novice programmers. Several solution have been proposed to overcome the challenges in programming education, include: pair-programming, software visualization (SV), automated assessment tools, programming environments and debuggers, interactive e-books, and enhanced IDE (4). These technologies promise to improve student programming skills, especially using visualization. Algorithm visualization (AV), and program visualization (PV) are two main types of SV. The main goal of PV is to display the runtime behaviour and execution of the program in order to help the student understand how the program is executed in the background. The usage of visualization to support teaching and learning of programming begun in 1980s (5). PV could be a

valuable resource to help novice students to improve their programming skills, and build a correct mental model for program or algorithm execution. Disappointingly, the pedagogical effectiveness of PV has shown mixed results (6); hence, the effectiveness of PV remains an open issue.

Student engagement and collaboration are important factors in enhancing the effectiveness of PV (7); as such, these factors must be considered carefully when designing a PV system. Engagement Taxonomy (ET) and Extended Engagement Taxonomy (EET) have been proposed as guidelines as to how to increase student engagement with software visualization (5,7). Furthermore, in software visualization research, the role of engagement in SV has received increased attention over the last decade and has been influenced by the works of (8), and (5). The development concept has shifted the focus toward engagement in terms of how to increase engagement and active learning when both designing and evaluating new tools (9,10). Student engagement is correlated with several positive indicators with PV, such as increased learning time, time-on-task, student motivation and student retention. As a result, researchers have been paying more attention on how to engage learners with PV, or with educational technologies in general, however, it has been shown that existing PVs have failed to engage students effectively (11).

On the other hand, collaborative learning has been actively implemented in computer science education, such as is pair programming. (7) stated that establishing a relationship between the engagement level and collaboration process is essential. As the levels of engagement increased, it increases the opportunity to improve collaborative activities; as a result, instructors need to figure out ways to use PV at these levels (7). (12) states that collaborative learning will increase the spent time by a student in solving exercise individually. Possible forms of collaboration in PV include writing code collaboratively, running it, and exploring a step-by-step, embedded textual chat in the system. Codechella, and Villa have provided good examples of collaborative PV system implementation features. Nevertheless, discussion about how PV enhances student learning with the potential benefits of the collaboration is still scarce (12).

Finally, designing a low cognitive load activity is important to help novices, while it is also important to increase the benefit from engaging in collaborative activities, particularly in terms of gained knowledge. Worked examples and Parson's problems are two examples of activity with a low cognitive load. Worked examples have been proven to have a positive effect on beginners when acquiring their first skills (13).

In this paper, the author proposes a new technique to improve student engagement and learning outcomes with PV. Further, it sets out to explore how to improve student engagement with PV using social worked-example techniques. This technique is based on the sociocultural theory and cognitive load theory. A detailed discussion of the theoretical background is presented in the following section. In the third section, the new technique is presented and discussed in detail. The final section draws together the key findings and the future work.

Theoretical Background:

Constructivism

Previous studies have observed the lack of discussion about the theoretical framework that lies behind the design and development of PV (14–16). The lack of theory and framework in the development of artefacts has long been argued in the literature on CER and educational technology (17,18). Learning about the relevant theories is a good starting point in understanding the learning process (19). Constructivism has a long history in cognitive psychology, while it also has a place in computer science (CS) education (14,17,20). This view is supported by (21), who argue that the combination of constructivism learning theory with the use of technology could make use of the best application of educational technology, while it would also facilitate the course design.

Moreover, constructivism and active learning theory are commonly used in the PV domain, as supported by the findings of (8) and (5), which suggest that active approaches are more effective. Constructivism states that people actively construct knowledge rather than passively receive and store ready-made knowledge. On constructivism, the student-centred is the main pedagogy that emphasis more on the student's previous experience rather than the teacher's, and on the active construction of knowledge rather than the passive receipt of information. Constructivism has many interpretations; among them, Piaget's cognitive constructivism and Vygotsky's social constructivism.

A cognitive constructivist theory predicted that the more effort put in to engage students in an activity, the more robust learning could be achieved

(8,17). In light of cognitive constructivism, the role of SV is not as an artefact to transfer knowledge to the student, but rather to enable the student to construct the knowledge through active engagement (8). Active learning is one of the principles of cognitive constructivism, whereby the learner could actively construct new understandings by becoming actively engaged with their activity (22). Therefore, the PV designer should consider different types of activities and engagement features to increase the learners' active engagement and enable them to construct knowledge (8,20).

Social constructivism

Social constructivism, which is influenced by Vygotsky's sociocultural theory, places a greater emphasis on the social context. This theory indicates that knowledge construction is stimulates from learner's feedback interaction. As the process of creating knowledge cannot be isolated from the social environment, learning is thus viewed as a process of active knowledge construction. This theory perspectives on learning argue that cognitive development is a social process rather than an individual process. Social constructivism has variety of theories, includes: Vygotsky's sociocultural theory (SCT) (23), Piaget's socio cognitive conflict theory (24) and Bandura's social cognitive theory (25).

Social constructivists focus on the important role of social and cultural nature of individuals' knowledge construction and tend to see knowledge as something that is defined through social collaboration and language use. In addition, the theory asserts that cognitive development depends on social interaction from guided learning. In PV, the largest focus on the development was more toward personal constructivism. Based on Piaget's cognitive constructivism, personal constructivism emphasizes the construction of knowledge by individuals. In contrast, social constructivists emphasize the role of the social and cultural nature in the construction of knowledge. In spite of Vygotsky's social constructivism, program visualization is seen as a sociocultural tool, as pointed out by (14).

Cognitive Load Theory

Cognitive load theory (CLT) is important due to the complex nature of learning programming languages. CLT has been conceived as a form of guidance for instructional designers eager to create instructional resources that are presented in a way that encourages the activities of the learners and optimizes their performance, and in turn, their learning (26). The theory of cognitive load provides an explanation as to why learning is impaired due to the exceeding limitation of working space capacity. Cognitive load is defined as the effort needed to manage the flow of information during instruction (27). The theory distinguishes two different types of cognitive load on a student's working memory: intrinsic load and extraneous load (28,29). CLT emphasises the role of working memory during learning due to its limited capacity. For example, during the learning process, if the working memory is overloaded, this could exhaust the learner.

For novice programmers, programming considered to have a highly intrinsic cognitive load since it simultaneously recalls different concepts, constructs and language syntax (30). Likewise, in the domain of computer programming, the extraneous cognitive load is high and is caused by programming language itself and the development tools (30). (31) illustrates this point clearly, as they argue that "From the first line of a Java program, you know we are in serious trouble: *public static void main (String[ ] args)* We have visibility modifiers, return types, method names, a class, parameters and arrays, and we haven't started the program". Several different concepts appear in just one line, which makes it harder to novice to connect them all; as a result, the working memory gets overloaded fast. This issue occurs in many other programming languages, and even in new languages that have introduced a very simple syntax (e.g., Python) because the student still needs to master a multitude of new concepts and techniques.

There are many applications of CLT in the domain, such as worked-example and Parson's problem. These activities could provide the entire solution to a problem, which the learner can study, and completion problems, which provide partial solutions for learners to fill in. The principle of the worked example (or worked-out example) is derived from the cognitive load theory, which refers to a step-by-step solution to a problem (32). According to CLT, for a novice student, using worked examples will reduce the cognitive load placed on learners to learn new concepts. When lecturer asks novice student to solve a problem individually in early learning phases that could make them exhausted, that could strain their working memory; however, it is recommended to teach student step-by-step tutorial, first, of how an expert solves those problems (worked examples). Recent studies have shown that worked-examples improve learning (33), and student engagement (34).

Another feature driven from CLT are Parson's problems, which focus on reducing the cognitive load in novice students when learning to program. Parson's problems are used to reduce the cognitive load of an activity, which is a type of code completion problem. It has also been named as Parson's puzzles, Parson's programming puzzles and Mangled code (33). In Parson's problems, a correct code is fragmented and mixed in several code blocks in which the student has to piece the blocks together to regenerate the correct code. It could be useful to teach syntactic and semantic language constructs (33). Parson's problems are designed as an engaging programming practice, which did not require students to type any code or encounter syntax errors (35), as shown in Figure 1. In Parson's problems, the design should return instant feedback by highlighting blocks that are in the wrong place or have the incorrect indentation (35). Using Parson's problems could effectively enhance novice code writing skills (34,36), because it only requires a student to understand the problem, and it never produces a syntax error.

Figure 1. Parsons problem exercise from Runestone e-book.

Related Work on PV:

Several PV systems were reviewed concerning to the aforementioned design principles. As shown in Table 1, four existing PV systems were compared in terms of their features. First, Runestone is an interactive e-book that teaches computer science to novice learners (27,33,35). The e-book includes a PV system to visualize the code execution for a student while they are going through the e-book. CLT is imperative in designing this e-book. Parson's problems and worked-examples are used heavily in the e-book in the form of the Example + Practice approach. This approach has a lower cognitive load. This research investigates a number of design principles based on CLT, however, the Runestone e-book is poor in terms of its social features, which, based on our theoretical framework, are important to the learning process.

Table 1 Comparison of existing tools.

PV	Worked-example	Parsons problem	Control flow	Narration	Discussion	Chat
Runestone (24,30,32)	X	X	X			
Codepourri (34)	X		X	X		X
Codechella (35)	X		X			X
ViLLE (36)	X		X	X		

Note: (X) indicates the feature was implemented

Online Python Tutor is a web-based PV, which is not limited to Python only; it supports six other programming languages: Java, JavaScript, TypeScript, Ruby, C and C++. It has become a popular PV tool for CS education and allows students to step through the code execution and visualize the state during that. Nevertheless, this tool does not address the context of the social interaction. However, several tools have developed on top of the python tutor to solve this issue, such as Codepourri, and Codechella. Codepourri extends Python Tutor to visualize students' annotated

worked examples (37). Figure 2 shows Codepourri utilizing annotation to provide line-by-line explanations for worked examples. In doing so, it crowdsources the process of adding annotations to any line of code. In addition, it enables students to vote the best annotation in order to be used in tutorial creation. On the other hand, Codechella is built upon Python Tutor; instead, it adds the real-time collaborative code writing to Python Tutor (37). In addition, it adds a chat feature, multiple mouse cursor sharing, and executes the visualization together. This study reported an improvement in student engagement and knowledge acquisition.

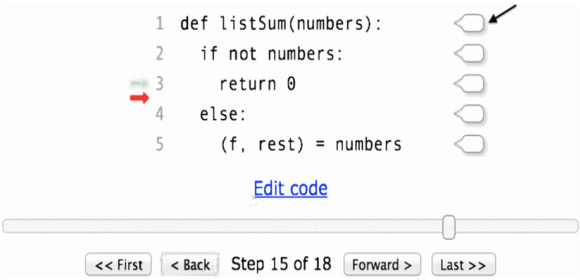

Figure 2. Codepourri add annotation to worked example (38).

Finally, ViLLE is a program visualization tool designed for teaching basic programming to novice programmers. It was first developed in 2007 and continues to evolve by introducing new features (39). Currently, ViLLE has now become a collaborative tool that enables students to work collaboratively to solve assignments and earn rewards as a group. The platform enables students to discuss and chat while they do their assignments. Thus far, several studies have found ViLLE is beneficial in learning fundamental programming and that the collaborative use of the tool improves learning even more. Figure 3 presents the main feature of ViLLE, which includes control flow, visualizing the state of a program, and programming line explanation (as a narration).

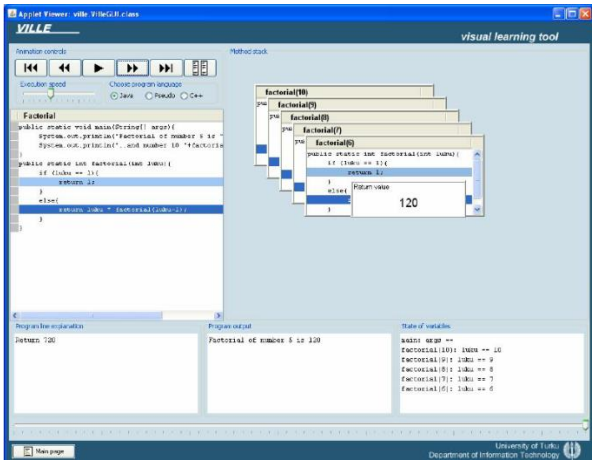

Figure 3. ViLLE user interface (40).

The Proposed Social Worked-Example (SWE) Technique:

This paper proposes a technique to enhance learning effectiveness and student engagement, based on the aforementioned theoretical framework. The social worked-example Technique (SWE) is a technique that transforms the traditional worked-example into a social activity, whereby a greater focus is placed on the collaborative role in constructing students' knowledge. The SWE technique has been introduced as an interactive activity to engage the student in using PV. This technique is based on a number of attractive principles that include low-load activity, narrative interaction and social collaboration.

Low-load activity. According to CLT, for a novice student, using worked examples will reduce the cognitive load placed on them to learn new concepts. However, there is little empirical evidence of worked examples in the programming education domain (34). Low cognitive load activities could lead to effective and engaging learning (27). The PV activity should maintain the student's optimal working-memory to avoid overloading the working memory. In contrast, using a poor design for this activity will lead to overloading the working memory too early, which negatively impacts the student engagement and learning process.

For the novice student, it is recommended to define or extend programming code rather than write it from scratch. It has been suggested that each example is followed by one or two practice activists (35). As consequence, each worked-example will be followed by a Parson's problem or more to focus attention on worked example. Parson's problems provide feedback by highlighting blocks that are in the wrong place or

have the incorrect indentation. Obtaining prompt feedback as to whether the answer is correct is an important factor to consider when designing Parson's problems. This could be in terms of highlighting the wrong step and/or providing more description or hints to accomplish the problem. Finally, a distractor, which are extra blocks that are not needed for correct answers could be used with Parson's problems.

Narrative interaction. Narrative contents and textual explanations are important factors that are often used in program visualization to help the student to better understand the concept explained within the system (41). Moreover, it could also help a student to understand the graphical representation of the execution of the code, which is represented visually (42). Integrating a narrative in PV to explain each step of the executed code may enhance the student's understanding of the executed code. In addition, it will also help teachers to highlight an important topic while the code is executed. Furthermore, collaboration contributes positively to student engagement with visualization.

In the SWE, the system will display a unique narration of each step of the code execution process. When writing a worked-example, the teacher is also required to provide either a description or explanation of each phase of execution. The narrative could either discuss what is happening right now or why it is happening. Such explanations are important for students to gain insight and a deep understanding of code execution. As PV has a control flow, the student can either go back or forward within the PV to traverse the narration provided.

Social collaboration. Social interaction is crucial to the learning process and is based on social constructivism. Since knowledge construction cannot be separated from the social environment, considering PV as a collaborative learning environment is an interesting perspective (7). Visualization provides a shared external representation to the different peers (7).

Combining PV and collaborative activities requires a careful design decision for success. In SWE, we integrate collaborative activities in several ways. First, for each narration added by the lecturer, the student has the ability to comment, discuss or ask questions regarding the specific step. Teachers or other students are allowed to contribute to the discussion or answer the questions. This feature is important to both student and teachers in different ways, while it also creates a channel between students and teachers in discussing PV. From the teacher's perspective, this kind of discussion could provide a number of useful insights, such as determining which step is ambiguous for the student, understanding how students are thinking, identifying student misconceptions, etc. From a student's perspective, PV could enhance social interaction, which would thus result in better learning. The online discussion brings many benefits to students, such as building a learning community, facilitating knowledge sharing, enhancing student engagement, and encouraging high-level thinking (43).

Conclusion:

This paper investigated the theoretical background as to how to improve worked example techniques in order to enhance student engagement. Considering constructivism, social constructivism and cognitive load theories, we propose a social worked-example technique. SWE is a technique that aims to turn worked-examples into a social activity by placing a greater focus on the collaborative role in constructing knowledge. As such, the proposed technique suggests several principles which are: low-load activity, narrative interaction and social collaboration; that can be applied in order to gain desired outcomes. Further experimental investigations are needed to evaluate this technique and measure its effect on student grade, programming skills and student engagement.

Acknowledgments:

This work was supported by the Ministry of Higher Education of Malaysia under Fundamental Research Grant Scheme (FRGS) [S/O code: 13581].

Conflicts of Interest: None.

References:

1. Alhazbi S. Active blended learning to improve students' motivation in computer programming courses: A case study. In Advances in engineering education in the Middle East and North Africa 2016 (pp. 187-204). Springer, Cham.
2. Konecki M, Kadoić N. Intelligent assistant for helping students to learn programming. In2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) 2015 May 25 (pp. 924-928). IEEE.
3. Drosos I, Guo PJ, Parnin C. HappyFace: Identifying and predicting frustrating obstacles for learning programming at scale. In2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) 2017 Oct 11 (pp. 171-179). IEEE.
4. Pears A, Seidman S, Malmi L, Mannila L, Adams E,

Bennedsen J, Devlin M, Paterson J. A survey of literature on the teaching of introductory programming. InACM sigcse bulletin 2007 Dec 1 (Vol. 39, No. 4, pp. 204-223). ACM.
5. Naps T, Rodger S, Velázquez-Iturbide Á, Rößling G, Almstrum V, Dann W, et al. Exploring the role of visualization and engagement in computer science education. SIGCSE Bull. 2003 Jun;35(2):131.
6. Ben-Ari M, Bednarik R, Levy RB-B, Ebel G, Moreno A, Myller N, et al. A decade of research and development on program animation: The Jeliot experience. J Vis Lang Comput. 2011;22(5):375–84.
7. Myller N, Bednarik R, Sutinen E, Ben-Ari M. Extending the Engagement Taxonomy: Software Visualization and Collaborative Learning. Trans Comput Educ. 2009;9(1):7:1-7:27.
8. Hundhausen C, Douglas SA, Stasko JT. A Meta-Study of Algorithm Visualization Effectiveness. J Vis Lang Comput. 2002;13(3):259–90.
9. Sorva J, Karavirta V, Malmi L. A Review of Generic Program Visualization Systems for Introductory Programming Education. ACM Trans Comput Educ. 2013;13(4):15.1-15.64.
10. Yohannis A, Prabowo Y. Sort Attack: Visualization and Gamification of Sorting Algorithm Learning. In: 2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games). IEEE; 2015. p. 1–8.
11. Végh L, Takáč O. Using Interactive Card Animations for Understanding of the Essential Aspects of Non-recursive Sorting Algorithms. In: Janech J, Kostolny J, Gratkowski T, editors. Proceedings of the 2015 Federated Conference on Software Development and Object Technologies. Cham: Springer International Publishing; 2017. p. 336–47.
12. Rajala T, Kaila E, Holvitie J, Haavisto R, Laakso M-J, Salakoski T. Comparing the collaborative and independent viewing of program visualizations. In: 2011 Frontiers in Education Conference (FIE). IEEE; 2011. p. F3G–1–F3G–7.
13. Atkinson RK, Derry SJ, Renkl A, Wortham D. Learning from Examples: Instructional Principles from the Worked Examples Research. Rev Educ Res. 2000 Jun;70(2):181–214.
14. Hidalgo-Céspedes J, Marín-Raventós G, Lara-Villagrán V. Learning principles in program visualizations: A systematic literature review. In: 2016 IEEE Frontiers in Education Conference (FIE). 2016. p. 1–9.
15. Velázquez-Iturbide Á, Hernán-Losada I, Paredes-Velasco M. Evaluating the Effect of Program Visualization on Student Motivation. IEEE Trans Educ. 2017;PP(99):1–8.
16. Shaffer C, Cooper M, Alon AJD, Akbar M, Stewart M, Ponce S, et al. Algorithm Visualization: The State of the Field. ACM Trans Comput Educ. 2010 Aug;10(3):1–22.
17. Malmi L, Sheard J, Simon, Bednarik R, Helminen J, Kinnunen P, et al. Theoretical Underpinnings of Computing Education Research: What is the Evidence? In: Proceedings of the Tenth Annual Conference on International Computing Education Research. New York, NY, USA: ACM; 2014. p. 27–34. (ICER '14).
18. Sheard J, Simon S, Hamilton M, Lönnberg J. Analysis of research into the teaching and learning of programming. InProceedings of the fifth international workshop on Computing education research workshop 2009 Aug 10 (pp. 93-104). ACM.
19. Meyer KA, editor. Student Engagement Online: What Works and Why: ASHE Higher Education Report, Volume 40, Number 6. John Wiley & Sons; 2014 Dec 4.
20. Moreno A, Sutinen E, Joy M. Defining and evaluating conflictive animations for programming education: The case of Jeliot ConAn. InProceedings of the 45th ACM technical symposium on Computer science education 2014 Mar 5 (pp. 629-634). ACM.
21. Gilakjani AP, Lai-Mei L, Ismail HN. Teachers' use of technology and constructivism. International Journal of Modern Education and Computer Science. 2013 May 1;5(4):49.
22. Urquiza-Fuentes J, Velázquez-Iturbide JÁ. Toward the effective use of educational program animations: The roles of student's engagement and topic complexity. Computers & Education. 2013 Sep 1;67:178-92
23. Vygotsky LS. 1978 Mind in Society (Cambridge, MA: Harvard University Press).
24. Piaget J. The equilibration of cognitive structures: The central problem of intellectual development. University of Chicago Press; 1985.
25. 25. Bandura, A. Social foundations of thought and action: A social cognitive theory. Englewood Cliffs, NJ: Prentice- Hall, Inc. 1986.
26. Chandler P, Sweller J. Cognitive Load Theory and the Format of Instruction. Cogn Instr. 1991 Dec;8(4):293–332.
27. Ericson B, Guzdial M, Morrison B. Analysis of Interactive Features Designed to Enhance Learning in an Ebook. In: Proceedings of the Eleventh Annual International Conference on International Computing Education Research. New York, NY, USA: ACM; 2015. p. 169–78. (ICER '15).
28. Morrison BB, Margulieux LE, Ericson B, Guzdial M. Subgoals Help Students Solve Parsons Problems. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16. New York, New York, USA: ACM Press; 2016. p. 42–7.
29. Morrison BB, Dorn B, Guzdial M. Measuring cognitive load in introductory CS. In: Proceedings of the tenth annual conference on International computing education research - ICER '14. New York, New York, USA: ACM Press; 2014. p. 131–8.
30. Mason R, Cooper G. Mindstorms robots and the application of cognitive load theory in introductory programming. Comput Sci Educ. 2013

Dec;23(4):296–314.
31. Bailie F, Courtney M, Murray K, Schiaffino R, Tuohy S. Objects First - Does It Work? J Comput Sci Coll. 2003 Dec;19(2):303–5.
32. Sweller J, Ayres P, Kalyuga S. The Worked Example and Problem Completion Effects. In: Cognitive Load Theory. New York, NY: Springer New York; 2011. p. 99–109.
33. Ericson B, Margulieux L, Rick J. Solving parsons problems versus fixing and writing code. In: Proceedings of the 17th Koli Calling Conference on Computing Education Research - Koli Calling '17. New York, New York, USA: ACM Press; 2017. p. 20–9.
34. Sorva J, Seppälä O. Research-based Design of the First Weeks of CS1. In: Proceedings of the 14th Koli Calling International Conference on Computing Education Research. New York, NY, USA: ACM; 2014. p. 71–80. (Koli Calling '14).
35. Ericson B, Rogers K, Parker M, Morrison B, Guzdial M. Identifying Design Principles for CS Teacher Ebooks through Design-Based Research. In: Proceedings of the 2016 ACM Conference on International Computing Education Research - ICER '16. New York, New York, USA: ACM Press; 2016. p. 191–200.
36. Sirkiä T. Combining parson's problems with program visualization in CS1 context. In: Proceedings of the 16th Koli Calling International Conference on Computing Education Research - Koli Calling '16. New York, New York, USA: ACM Press; 2016. p. 155–9.
37. Guo PJ, White J, Zanelatto R. Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning. In: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE; 2015. p. 79–87.
38. Gordon M, Guo PJ. Codepourri: Creating visual coding tutorials using a volunteer crowd of learners. In: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE; 2015. p. 13–21.
39. Laakso M-J, Kaila E, Rajala T. ViLLE – collaborative education tool: Designing and utilizing an exercise-based learning environment. Educ Inf Technol. 2018 Jul;23(4):1655–76.
40. Rajala T, Laakso M-J, Kaila E, Salakoski T. VILLE – Multilanguage Tool for Teaching Novice Programming. Turku Centre for Computer Science; 2007. (TUCS Technical Reports).
41. Urquiza-Fuentes J, Velázquez-Iturbide JÁ. Pedagogical Effectiveness of Engagement Levels – A Survey of Successful Experiences. Electron Notes Theor Comput Sci. 2009 Jan;224:169–78.
42. Urquiza-Fuentes J, Velázquez-Iturbide Á. A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems. Trans Comput Educ. 2009;9(2):9:1-9:21.
43. Ding L, Er E, Orey M. An exploratory study of student engagement in gamified online discussions. Comput Educ. 2018;120(February):213–26.

تقنيات العمل الاجتماعي امثلة - لتعزيز مشاركة الطلاب في برنامج التصور

عبدالله السخاف **مازن عمر** **مازدة احمد**

جامعة اوتارا ماليزيا، ماليزيا

الخلاصة:

يعد تعلم البرمجة من بين أهم التحديات في تعليم علوم الكمبيوتر. حاليا، يتم استخدام تصوير البرامج (PV) كأداة للتغلب على معدلات الفشل والتسرب العالية في مادة اساسيات البرمجة. ومع ذلك، هناك مخاوف متزايدة بشأن فعالية أدوات تصوير البرامج الحالية استنادا الى النتائج المختلطة المستمدة من الدراسات المختلفة. تعتبر مشاركة الطلاب أيضًا عاملاً حيويًا في بناء PV ناجحًا، كما تعد أيضًا جزءًا مهمًا من عملية التعلم بشكل عام. تم إدخال العديد من التقنيات لتعزيز المشاركة في أدوات تصوير البرامج؛ ومع ذلك، فإن مشاركة الطلاب في PV لا يزال يمثل تحديًا كبيراً. استخدمت هذه الورقة ثلاث نظريات مختلفة: البنيوية، والبناء الاجتماعي، والحمل المعرفي لاقتراح تقنية لتعزيز مشاركة الطلاب في استخدام أدوات تصوير البرامج. تعمل تقنية الأمثلة المكتملة الاجتماعية (SWE) على تحويل المثال المكتمل التقليدي إلى نشاط اجتماعي ، حيث يتم التركيز بشكل أكبر على دور التعاون في بناء معرفة الطلاب. حددت هذه الدراسة ثلاثة مبادئ يمكن أن تعزز مشاركة الطلاب من خلال تقنية SWE: التعلم النشط والتعاون الاجتماعي والأنشطة ذاتس التحميل المنخفض.

الكلمات المفتاحية: تصوير البرامج، تعليم البرمجة التمهيدية،