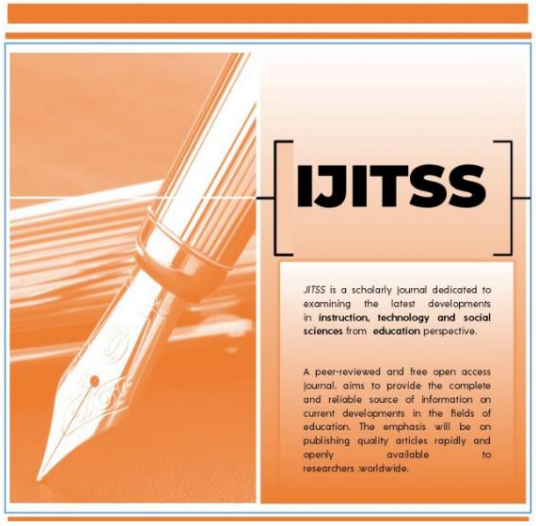
e-ISSN: 2716-6546

International Journal of Instruction, Technology & Social Sciences

International Journal of Instruction, Technology & Social Sciences
www.ijitsc.net



ISSN: 2716-6546

Design of Worked Examples for Learning Programming: Literature Review

Mariam Nainan[1], Balamuralithara Balakrishnan[2], Ahmad Zamzuri Mohamad Ali[3]

[1]*Sultan Idris Education University*
[2]*Sultan Idris Education University*
[3]*Sultan Idris Education University*

To cite this article:

Nainan, M., Balakrishnan, B., & Mohamad Ali, A.Z. (2020). Design of worked examples for learning programming: Literature review. *International Journal of Instruction, Technology, and Social Sciences (IJITSS), 1*(3), 8-16.

Design of Worked Examples for Learning Programming: Literature Review

Mariam Nainan, Balamuralithara Balakrishnan, & Ahmad Zamzuri Mohamad Ali

Article Info

Article History

Received:
15 October 2020

Accepted:
5 December 2020

Keywords

Worked examples,
Example-based learning,
Programming education

Abstract

Learning from worked examples, or example-based learning, has been found to be effective for learning problem solving. Several instructional principles for worked example design have been proposed based on research studies conducted in several domains. However, research on the design of worked examples for programming education is limited. This study reviews research studies on worked examples proposed for teaching and learning how to solve programming problems and analyses the proposed designs with respect to the instructional principles for example-based learning. This paper presents the results of the analysis and the characteristics of the proposed designs. This paper also discusses the findings and suggests areas for further research.

Introduction

Learning from worked examples, or example-based learning is a strategy that has been proposed as an alternative to learning through problem solving (Hoogerheide, & Roelle, 2020; Renkl, 2014; 2017; Sweller & Cooper, 1985; Van Gog et al., 2019). A worked example contains a problem specification and a solution (Renkl, 2014). It demonstrates how an expert would solve the given problem. Example-based learning has been found to be effective for beginners or novice problem solvers who are newly introduced to domain concepts or principles during their initial acquisition of problem solving skills (Hoogerheide, & Roelle, 2020; Renkl, 2014; 2017; Van Gog et al., 2019). Based on studies on effectiveness of different worked examples designs, Renkl (2014) proposed a theory of example-based learning and instructional principles for worked example design.

Although example-based learning has been widely researched in other educational domains (e.g., recent studies in engineering (Dart et al., 2020), mathematics (Hesser & Gregory, 2015; Retnowati, 2017), physics (Badeau et al., 2017; Saw, 2017), this has not been the case for programming education (Skudder & Luxton-Reilly, 2014), although there has been increasing interest in recent years. The aims of the current study were to review research studies which proposed different worked example designs for the programming domain, analyse them in relation to Renkl's (2014) instructional principles, and describe characteristics of the proposed designs. This paper presents results of the analysis, discusses the results, and suggests areas for further research before concluding.

Instructional Principles for Example-Based Learning

Atkinson et al. (2000) proposed instructional principles for worked example design based on research studies in various domains although, for some studies, the domain was not mentioned. These principles were: integrate information in multiple representations, demarcate subgoals, use multiple examples of the same or varying problem types, and encourage self-explanation. Van Gog and Rummel (2010) suggested a similar list of principles and added providing explanations, use of erroneous worked examples, and imagination or cognitive rehearsal whereby learners mentally rehearse what were illustrated in the examples. Another principle relevant for examples in observational learning was the model-observer similarity principle.

Renkl (2014) proposed a theory of example-based learning which was constructed from a synthesis of research studies in three different theoretical areas, namely, learning from worked examples, analogical reasoning, and observational learning. He derived a comprehensive list of instructional principles from results of empirical studies which had produced positive effects and were conducted for different education domains, although some of the principles were well researched only for mathematics domain. The following subsections present Renkl's list of principles. These principles encompassed those proposed by Atkinson et al. (2000) and Van Gog and Rummel (2010) as well. They are discussed in general, that is, not for a specific domain.

Self-Explanation Principle

To benefit from example-based learning, learners must explain to themselves how the solution or solution steps presented in the examples relate to domain principles, concepts, or operators. Learners should be prompted to self-explain to improve their learning outcomes.

Comparison Principle

Learners should be given multiple examples that apply the same principles or concepts and should be prompted to compare them. The intention is that learners abstract principles and concepts from the specific details given in the examples. They should realise the structural-relevant features and ignore superficial details.

Explanation-Help Principle

When learners are not able to generate explanations or abstract principles themselves, explanations may be added to the examples instead.

Model-Observer Similarity Principle

With respect to observational learning, this principle states that the model demonstrating the tasks to be learnt should be similar to the learner in certain aspects, such as age.

Example-Set Principle

Examples should be presented as a set of two or more. The examples in the set may present problems with the same structural features but with different cover stories, similar cover stories but different types of problems, or alternative solutions to the same problem.

Easy-Mapping Principle

When information is represented in multiple forms in an example, learners should be assisted in mapping related pieces of information in the different representations.

Meaningful Building Blocks Principle

When the solution presented in the example consists of multiple steps, related steps should be combined into meaningful building blocks and presented in a manner that is easily identifiable by the learner. Also, a complex step should be broken into smaller simpler steps.

Studying Errors Principle

Examples may include errors which are highlighted to learners. Such examples are useful to help learners avoid such errors themselves.

Imagery Principle

Learners should imagine solving the same problems given in the examples in their minds, without looking at the examples.

Interleaving by Fading Principle

A set of isomorphic examples should be presented where the second example would have a missing step in the solution which learners would have to complete. Subsequent examples would have more and more steps left out,

until finally, learners are presented with problems which they have to solve entirely on their own. This is called fading effect.

Existing Reviews of Research on Worked Examples for Programming Education

Caspersen and Bennedsen (2007) proposed the use of worked examples in the design of a programming course. They suggested guidelines similar to Renkl's (2014) meaningful building block, example-set, and self-explanation principles for the worked examples. In addition, they suggested that worked examples should be interleaved with problems. Skudder and Luxton-Reilly (2014) made suggestions on how worked examples could be used for computer science education based on a review of studies conducted for other domains. Their suggestions were aligned to Renkl's meaningful building block, fading, and self-explanation principles. They also reviewed studies that had been conducted specifically for programming education. These covered investigations of how worked examples should be presented: examples only, example-problem pairs, example-problem blocks, and examples with high and low variability. In addition, they stated that the common practice in other domains was to present worked examples interleaved with problems. Other studies reviewed applied the meaningful building block and fading principles.

Method for Current Review of Worked Example Designs for Programming Education

Unlike the review of Skudder and Luxton-Reilly (2014), research studies for review and analysis in this study were selected on the criteria that the worked examples proposed in those studies were designed specifically for programming education. Furthermore, the worked example designs were analysed according to Renkl's (2014) instructional principles since these principles have been well established from research studies in other domains. On another note, some research studies focus on the content of the example rather than its design (e.g., Harsley et al. (2016) conducted investigations on worked examples with three types of content: a program solution, an analogical solution, and a combination of analogical and program solutions). However, the focus of this study was on worked example design.

Many of the selected studies evaluated the proposed example designs empirically but a few made only suggestions which had not been evaluated yet. Some evaluations were conducted in classroom settings over the duration of a course but a few had been evaluated only in single experimental session. The inclusion of the studies in this review did not imply that the reviewed designs had been shown to be effective. The purpose of this review was to capture as broad a scope of the different worked example designs that had been proposed for programming education. Some of the researchers targeted learners at university level, and others at high school level. The programming languages used also varied, such as text-based or drag-and-drop (i.e., block-based). On the other hand, some did not utilise a programming language at all, but instead, made use of algorithmic representations, such as pseudocode.

An issue that arose during the analysis was the criteria for identifying a worked example. For programming education, a worked example contains a problem specification and a program solution. But, examples may also be presented with the program solution but not the problem specification; learners may then be asked to determine the problem that the program solves. In other words, they may be asked to work out the overall purpose or goal of the program. The intention is to develop learners' program comprehension ability, which contributes to their problem solving ability. Hence, in the current study, the selected studies were not limited to worked examples (with problem specifications and solutions) but also to examples without problem specifications. However, examples that contained explanations of individual program statements but did not prompt learners for the purpose or function of the entire program (e.g. (Hosseini et al., 2016)), were not selected. They were primarily intended to help learners understand programming language features, rather than how to solve problems or to work out the overall purpose of the given solutions. Therefore, the term example is used rather than worked example in the following sections.

Proposed Example Designs and their Characteristics

For the studies selected and reviewed in this study, the example designs proposed were analysed to determine which of Renkl's (2014) instructional principles the designs were aligned to. It is noted that certain example designs were aligned to more than one instructional principle, and some researchers proposed multiple types of example designs. Table 1 lists the instructional principles and the studies employing those principles for their example designs.

Table 1. Instructional Principles and Studies which Employed Them

Instructional Principles	Reviewed Studies
Self-Explanation	Alhassan (2017); Ichinco & Kelleher (2015); Kumar (2016); Moura (2012; 2013); Patitsas et al. (2013); Vieira et al. (2015); Vihavainen et al. (2015); Zhi et al.(2019)
Comparison	Lui et al. (2008); Patitsas et al. (2013)
Explanation-Help	Hohn & Moraes (1997); Hosseini et al. (2018); Vieira et al. (2015)
Example-Set	Tepgeç & Cevik (2018); Vieira et al. (2015)
Easy-Mapping	Hosseini et al. (2018); Vieira et al. (2015)
Meaningful Building Blocks	Hosseini et al. (2018); Margulieux et al. (2016); Morrison et al. (2015); Vieira et al. (2015)
Studying Errors	Moura (2012; 2013); Vihavainen et al. (2015)
Interleaving By Fading	Gray et al. (2007); Moura (2012; 2013)

The following subsections describe the characteristics of the example designs in the selected studies in relation to Renkl's (2014) instructional principles.

Self-Explanation Principle

To foster self-explanation of the program solution, the approach used by Alhassan (2017) was to ask learners to write down their explanations. Learners were told that the explanations would be assessed. Vihavainen et al. (2015) also used a similar approach, where learners were asked to explain the solutions in the examples. However, the learners were told that their explanations would not be assessed. Instead, they would be given points for attempting their best. These were some ways learners were given incentives to write down their explanations. Also, Patitsas et al. (2013) inserted questions in examples to prompt self-explanation but the researchers did not mention whether the explanations were assessed. In addition to open-ended questions, Vihavainen et al. (2015) also examined the effect of inserting multiple-choice questions to guide learners in developing their explanations.

The examples designed by Moura (2012) contained algorithmic (pseudocode) solutions and corresponding programs. Learners were asked to summarise what the solution's function or purpose was. This is another mechanism to encourage self-explanation. As an extension to the study, Moura (2013) introduced the use of a visualisation tool that learners could use to run the pseudocode solutions. The tool simulated the execution of the solution. It was intended to help learners obtain a better understanding of the solution, and so, guide them in self-explanation. Vieira et al. (2015) presented learners with two examples. The program in the first example was fully commented to describe the functions for the different parts of the programs. In the second example, comments were left out and learners were required to write comments for the program as part of their assignments. This was another form of self-explanation prompt. The researchers mentioned that learners' comments were assessed.

In another vein of research, Kumar (2016) used an intelligent tutoring system where learners were first asked to analyse given programs. Learners were guided in their analysis through questions. When a learner failed to answer the questions correctly, the system presented an example with the same task together with a correct analysis as a feedback. To encourage them to self-explain the feedback example, it contained questions. Learners answered these questions by selecting options from drop-down lists. After the learner had selected an answer, the system gave feedback on its correctness. In this manner, learners were prompted to self-explain and were given feedback on their explanations. Similarly, Zhi et al. (2019) used a computer-based programming environment in which learners were presented a problem and the program part by part. For each part, they were asked to answer questions by selecting options from drop-down lists. Feedback was given on the submitted answers. Once learners had answered the questions correctly, the next part of the program was presented. Likewise, Ichinco and Kelleher (2015) used a computer-based programming environment where learners could view examples during their problem solving activity. The example problem and program were presented part by part in correspondence to the problem to be solved. But, instead of prompting self-explanation through questions, the important concept in the program that learners should take note of was highlighted to encourage learners to self-explain.

Comparison Principle

In the study conducted by Patitsas et al. (2013), learners were presented with two examples which showed alternative solutions to same problem. The two conditions in their empirical study were to present the examples sequentially or in parallel. For both conditions, the examples contained questions to guide self-explanation. However, for the examples presented in parallel, these questions were supplemented with another question to compare the solutions as well. Again, to encourage learners to compare their solutions to alternatives, Lui et al. (2008) used learners' solutions

as examples. Learners were asked to share their solutions to given problems voluntarily by submitting them to a web-based system. Only learners who had shared their own solutions were allowed to view others' solutions. This restriction, in effect, acted as a filtering mechanism for solution quality, because more of the learners who shared their solutions were strong performers. This solution-sharing approach permitted learners to compare alternative ways to solve the same problem.

Explanation-Help Principle

Vieira et al. (2015) presented learners with examples that contained programs with comments as well as textual explanations which elaborated what the different parts of the program achieved. Hohn and Moraes (1997) designed examples which contained rule-based explanations, in the form of "if condition then action". The explanations elaborated on what actions to take in the overall program creation process and under what conditions the actions were to be taken. The examples designed by Hosseini et al. (2018) contained programs with comments and additional explanations on the purposes of different program statements. Further elaborations could be obtained on request.

Example-Set Principle

In the study conducted by Tepgeç and Cevik (2018), learners were presented with two examples for isomorphic problems, with same structural features but different surface details. Likewise, in the study conducted by Vieira et al. (2015), two examples were presented, both of which were illustrating a specific topic.

Easy-Mapping Principle

The examples used by Vieira et al. (2015) presented the solution in multiple representations (i.e., program, flowchart, and textual explanations). Corresponding elements in the different representations were labelled with the same numbers for easy mapping. Similarly, in the examples used by Hosseini et al. (2018), explanations for different program statements were mapped to those statements for ease of reference.

Meaningful Building Blocks Principle

In the examples designed by Vieira et al. (2015), statements in the program were grouped and comments were placed for each group to explain the purpose of that group. In a similar manner, program statements in examples used by Hosseini et al. (2018) and Morrison et al. (2015) were grouped and additional text was added as labels to explain the purpose of the group. Morrison et al.'s empirical study compared three example designs: not labelled, grouped and labelled with explanations, or grouped and labelled with placeholders for which learners had to generate meaningful labels. In all these studies, grouping and labelling were used to segment the program into meaningful sections.

Margulieux et al. (2016) designed examples in the form of textual instructions and video demonstrations. Their examples described sequence of steps to produce programs rather than presenting programs as solutions to problems. The steps, in both textual instructions and video demonstrations, were grouped and labelled. Hence, the grouping and labelling were used to segment the sequence of steps into meaningful sections.

Studying Errors Principle

In addition to examples containing correct solutions, Moura (2012) proposed examples in which the solutions contained errors. Learners were asked to find and correct the errors. In the second revision of the course conducted by Moura (2013), learners were asked to use a visualisation tool to help them make the corrections. Similarly, for some examples used by Vihavainen et al. (2015), the solution had errors. Learners were prompted to explain the cause of the errors and how to correct them.

Interleaving by Fading Principle

In addition to examples containing complete solutions, Moura (2012; 2013) also designed examples where the solutions had some missing sections. Learners were asked to complete the missing sections. Gray et al. (2007) also proposed the use of a set of examples with some missing or partially completed steps. As the learner progressed from one example to the next in the set, an increasing number of steps were left out. But, Gray et al. designed examples to illustrate the programming process rather than problems and their solutions. They specified the following dimensions of the process: design, implementation, semantics, execution, and verification. They also proposed that examples should illustrate programming language constructs as well, such as selection or repetition constructs. So, their examples illustrated language construct and programming dimension pairs, such as selection-design pair or selection-implementation pair.

Discussion

The aim of this study was to review research studies that proposed examples for programming education and analyse the example designs in accordance with Renkl's (2014) instructional principles for example-based learning. From the analysis, it is seen that, for the programming domain, examples were meant to depict either the solution for a problem (i.e., the program or corresponding representations) or the process of creating such a solution (i.e., the sequence of steps in the process). The aim was to support learners in understanding how programming problems are to be solved.

The instructional principle that was commonly found in the analysed designs was self-explanation principle. This is understandable since examples were effective only when learners engaged in self-explanations. By explaining the solution or the process to themselves, learners were expected to develop their problem solving knowledge in terms of program creation. This understanding will assist them in solving similar problems in the future. Different methods were used to guide self-explanation. The most common method was to include questions regarding the solutions given in the examples.

For some of the studies no feedback was given to learners on their answers. If learners do not obtain feedback, they are left to make their own judgements about their explanations (Vihavainen et al., 2015). In this situation, their lack of understanding or misunderstandings may not be adequately addressed. Feedback through manual grading of answers would be time-consuming for instructors and may not be timely enough to be useful to learners, unless they were multiple-choice questions. On the other hand, intelligent tutoring systems or computer-based programming environments provided automatic and immediate feedback. This is particularly helpful when learners require assistance in answering questions. Furthermore, learners are able to execute example programs in programming environments and see the programs' effects, which may further improve their understanding. Alternatively, the use of visualisation tools may offer additional assistance. However, ease of use of such tools has to be taken into consideration. Another possibility is to give learners the instructor's explanations and prompt them to compare to theirs. This is similar to the suggestion by Price et al. (2020) on providing feedback for programming homework.

When learners were asked to compare solutions, the examples included questions to prompt learners to make the comparisons. This was an application of the comparison principle. Just as for self-explanation questions, feedback to comparison questions may be necessary for effective learning. Closely related to the comparison principle is the example-set principle where a set of isomorphic or related examples were presented. Students need to engage in comparison processes to gain the benefits of studying these examples. To encourage self-explanations and comparisons, a possible method, other than question-and-feedback, is to provide training, through modelling and coaching (Renkl, 2014). So, a possible area of future research is to investigate how training could be provided to ensure that learners benefit when prompted to self-explain and compare.

Instead of prompting learners to self-explain, aligned to the explanation-help principle, explanations were given in some example designs. Explanations are necessary when learners do not have sufficient knowledge to explain on their own. There were variations in the level of detail of the explanations: each individual program statement level, groups of statements, or overall program. In order to assist learners to comprehend a solution at various levels of abstraction (i.e., individual statement level up to the overall program level), it may be beneficial if explanations are given at all levels. This may help learners develop abstraction ability which is important for solving programming problems (Luxton-Reilly et al., 2018). However, mapping of different levels of explanations to the relevant sections of the solution would be needed. The easy-mapping principle, which is useful when there are multiple representations of the solution, may be applicable here. Future research could investigate how examples can be designed to develop abstraction ability through the use of multiple levels of explanations.

The meaningful building blocks principle resulted in segmentation of the solution or the solution process into meaningful blocks. More specifically, the program (written in a text-based programming language) or the sequence of steps (for creating a program in a drag-and-drop based programming environment) were divided into segments. Dividing into segments helps learners to understand that the solution is made up of parts which achieve different purposes. A possible future area of research could be to investigate how learners could be assisted to generalise from the specific details in the different segments and to recognise patterns in terms of the essential elements and their structural relationships. Among the selected studies, Hohn and Moraes (1997) mentioned the use of programming patterns in their rule-based explanations. The concept of programming patterns and the importance of teaching patterns in programming education have been stressed by several researchers (e.g., (De Raadt et al., 2009; Muller & Haberman, 2008)). A major difference between expert and novice programmers is that experts have knowledge of programming patterns which results in their superior problem solving performance compared to novices (Robins et al., 2003). Experts' knowledge of patterns helps them solve problems effectively and efficiently (Luxton-Reilly et al., 2018).

Similar to other domains, the principle of studying errors is relevant in examples for programming education because finding and correcting errors are important tasks in the programming process. Likewise, just as for other domains, the principle of interleaving by fading is relevant for programming education to ease the transition from example study to problem solving. It is applicable for the examples which illustrate either the solution or the programming process.

It is noted that the model-observer similarity and imagery principles were not reflected in example designs among the analysed articles. Hence, this raises the question of their relevance in examples for programming education. However, the model-observer similarity principle may be relevant for live-coding (e.g., Raj et al., 2018) or live-streaming (e.g., Faas et al., 2018) which is modelled on cognitive apprenticeship (Collins et al., 1991).

The majority of the example designs proposed in the selected studies presented problems and their solutions. However, a few emphasised the programming process (e.g., (Gray et al., 2007). Although Gray et al. (2007) stated that the programming process began with problem analysis and design in the problem domain, they did not include analysis in their list of programming dimensions. On the other hand, in their description of the design dimension, there were elements which could be interpreted as belonging to the analysis dimension. Clarification of the tasks that should be performed during analysis as a separate dimension requires further investigation.

Conclusion

This paper presented an analysis of designs of worked examples proposed for use in programming education based on Renkl's (2014) instructional principles of example-based learning. From the studies reviewed, it is seen that most of the principles were reflected in the designs except for model-observer similarity and imagery principles. However, since evaluation of some of the proposed designs were limited or lacking and, because of the diversity in learners' backgrounds, varying learner age groups, as well as differences in text-based and block-based program development tools, more studies are required in order to establish characteristics of example designs that are applicable for the specific learning situations. Furthermore, just as for other domains, more research is needed to identify moderating factors for the effectiveness of worked examples for the programming domain.

Disclosure statement

No potential conflict of interest was reported by the authors.

Non-Funded

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

Alhassan, R. (2017). The effect of employing self-explanation strategy with worked examples on acquiring computer programming skills. *Journal of Education and Practice*, *8*(6), 186-196.

Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of Educational Research*, *70*(2), 181-214.

Badeau, R., White, D. R., Ibrahim, B., Ding, L., & Heckler, A. F. (2017). What works with worked examples: Extending self-explanation and analogical comparison to synthesis problems. *Physical Review Physics Education Research*, *13*(2), 1-27.

Caspersen, M. E., & Bennedsen, J. (2007, September). Instructional design of a programming course: A learning theoretic approach. In *Proceedings of the third international workshop on computing education research* (pp. 111-122). ACM.

Collins, A., Brown, J. S., & Holum, A. (1991). Cognitive apprenticeship: Making thinking visible. *American Educator*, *15*(3), 6-11.

Dart, S., Pickering, E., & Dawes, L. (2020). Worked example videos for blended learning in undergraduate engineering. *Advances in Engineering Education*, *8*(2), 1-22.

De Raadt, M., Watson, R., & Toleman, M. (2009). Teaching and assessing programming strategies explicitly. In *Proceedings of the 11th Australasian computing education conference (ACE 2009)*, *95*, 45-54). Australian Computer Society Inc..

Faas, T., Dombrowski, L., Young, A., & Miller, A. D. (2018). Watch me code: Programming mentorship communities on Twitch.tv. In *Proceedings of the ACM on human-computer interaction*, *2*(50), 1-18. ACM.

Gray, S., St. Clair, C., James, R., & Mead, J. (2007, September). Suggestions for graduated exposure to programming concepts using fading worked examples. In *Proceedings of the third international workshop on computing education research* (pp. 99-110).

Harsley, R., Green, N., Alizadeh, M., Acharya, S., Fossati, D., Di Eugenio, B., & AlZoubi, O. (2016, February). Incorporating analogies and worked out examples as pedagogical strategies in a computer science tutoring system. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 675-680). ACM.

Hesser, T. L., & Gregory, J. L. (2015). Exploring the use of faded worked examples as a problem solving approach for underprepared students. *Higher Education Studies*, *5*(6), 36-46.

Hohn, R. L., & Moraes, I. (1998). Use of rule-based elaboration of worked examples to promote the acquisition of programming plans. *Journal of Computer Information Systems*, *38*(2), 35-40.

Hoogerheide, V., & Roelle, J. (2020). Example-based learning: New theoretical perspectives and use-inspired advances to a contemporary instructional approach. *Applied Cognitive Psychology*, *34*(4), 787-792.

Hosseini, R., Akhuseyinoglu, K., Petersen, A., Schunn, C. D., & Brusilovsky, P. (2018, November). PCEX: interactive program construction examples for learning programming. In *Proceedings of the 18th Koli calling international conference on computing education research* (pp. 1-9).

Hosseini, R., Sirkiä, T., Guerra, J., Brusilovsky, P., & Malmi, L. (2016, February). Animated examples as practice content in a Java programming course. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 540-545). ACM.

Ichinco, M., & Kelleher, C. (2015, October). Exploring novice programmer example use. In *Proceedings of 2015 IEEE symposium on visual languages and human-centric computing (VL/HCC)* (pp. 63-71). IEEE.

Kumar, A. N. (2016, June). Using cloze procedure questions in worked examples in a programming tutor. In *Proceedings of international conference on intelligent tutoring systems* (pp. 416-422). Springer.

Lui, A. K., Cheung, Y. H., & Li, S. C. (2008). Leveraging students' programming laboratory work as worked examples. *ACM SIGCSE Bulletin*, *40*(2), 69-73.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018, July). Introductory programming: a systematic literature review. In *Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education* (pp. 55-106). ACM.

Margulieux, L. E., Catrambone, R., & Guzdial, M. (2016). Employing subgoals in computer programming education. *Computer Science Education*, *26*(1), 44-67.

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015, July). Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the eleventh annual international conference on international computing education research* (pp. 21-29). ACM.

Moura, I. C. (2012, July). Worked-out examples in a computer science introductory module. In *Proceedings of the world congress on engineering* (pp.1082-1085). Retrieved from http://www.iaeng.org/publication/WCE2012/WCE2012_pp1082-1085.pdf

Moura, I. C. (2013, July). Visualizing the execution of programming worked-out examples with Portugol. In *Proceedings of the world congress on engineering* (pp. 404-408). Retrieved from http://www.iaeng.org/publication/WCE2013/WCE2013_pp404-408.pdf

Muller, O., & Haberman, B. (2008). Supporting abstraction processes in problem solving through pattern-oriented instruction. *Computer Science Education, 18*(3), 187-212.

Patitsas, E., Craig, M., & Easterbrook, S. (2013, August). Comparing and contrasting different algorithms leads to increased student learning. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 145-152). ACM.

Price, T. W., Williams, J. J., Solyst, J., & Marwan, S. (2020, April). Engaging students with instructor solutions in online programming homework. In *Proceedings of the 2020 CHI conference on human factors in computing systems* (pp. 1-7).

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).

Renkl, A. (2014). Toward an instructionally oriented theory of example-based learning. *Cognitive Science, 38*(1), 1-37.

Renkl, A. (2017). Learning from worked-examples in mathematics: Students relate procedures to principles. *ZDM, 49*(4), 571-584.

Retnowati, E. (2017, April). Faded-example as a tool to acquire and automate mathematics knowledge. In *Journal of Physics: Conference Series* (Vol. 824, pp. 1-7).

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

Saw, K. G. (2017). Cognitive load theory and the use of worked examples as an instructional strategy in physics for distance learners: A preliminary study. *Turkish Online Journal of Distance Education, 18*(4), 142-159.

Skudder, B., & Luxton-Reilly, A. (2014). Worked examples in computer science. In *Proceedings of the 16th Australasian conference on computing education (ACE '14)* (Vol. 148, pp. 59–64). Australian Computer Society Inc..

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction, 2*(1), 59–89.

Tepgeç, M., & Çevik, Y. D. (2018). Comparison of three instructional strategies in teaching programming: restudying material, testing and worked example. *Journal of Learning and Teaching in Digital Age, 3*(2), 42-50.

Van Gog, T., & Rummel, N. (2010). Example-based learning: Integrating cognitive and social-cognitive research perspectives. *Educational Psychology Review, 22*(2), 155-174.

Van Gog, T., Rummel, N., & Renkl, A. (2019). Learning how to solve problems by studying examples. In J. Dunlosky & K. A. Rawson (Eds.), *The Cambridge Handbook of Cognition and Education* (p. 183–208). Cambridge University Press.

Vieira, C., Yan, J., & Magana, A. J. (2015). Exploring design characteristics of worked examples to support programming and algorithm design. *Journal of Computational Science Education, 6*(1), 2-15.

Vihavainen, A., Miller, C. S., & Settle, A. (2015, February). Benefits of self-explanation in introductory programming. In *Proceedings of the 46th ACM technical symposium on computer science education* (pp. 284-289). ACM.

Zhi, R., Price, T.W., Marwan, S., Milliken, A., Barnes, T. & Chi, M. (2019, February). Exploring the impact of worked examples in a novice programming environment. In *Proceedings of the 50th ACM technical symposium on computer science education*. (pp. 98-104). ACM.

Author Information

Mariam Nainan
Sultan Idris Education University
Tanjong Malim, Perak, Malaysia
Contact e-mail: *marnai2004@yahoo.com*

Balamuralithara Balakrishnan
Sultan Idris Education University
Tanjong Malim, Perak, Malaysia

Ahmad Zamzuri Mohamad Ali
Sultan Idris Education University
Tanjong Malim, Perak, Malaysia