Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level

Mariam Nainan [1], Balamuralithara Balakrishnan [2]

[1] Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Malaysia.
marnai2004@yahoo.com

[2] Faculty of Art, Computing & Creative Industry, Sultan Idris Education University, Malaysia.
balab@fskik.upsi.edu.my

http://dx.doi.org/10.17220/mojet.2019.04.003

ABSTRACT

Studying worked examples has been found to be effective for learning problem solving, especially among students. However, students need to actively process example content to benefit from it and content must be structured in a manner that facilities knowledge construction. This study investigated the use of worked examples for teaching and learning programming. Programming involves problem analysis and solution generation. But students tend to jump to solution generation without adequately analysing the problem. Consequently, the current study designed and implemented a new worked example design that emphasised problem analysis and utilised highlighting through web technology to encourage active processing of example content. This study also evaluated the new design in a quasi-experiment in a university course in Malaysia, compared to subgoal labelled worked examples, and conducted over three sessions. Posttest performance was analysed using independent samples t-test and frequency distributions. The results suggested that worked examples based on the new design were more effective than subgoal labelled worked examples, with statistically significant difference in performance, and medium effect size for the first session. For the second and third sessions, performance was marginally better, with learning in both groups possibly limited by the complexity of the worked examples and assessments.

Keywords: *worked examples, technology-supported learning, computer science education*

INTRODUCTION

Programming knowledge and skill are essential for computer science and software engineering students enrolled at tertiary education institutions. Nowadays, with the ubiquitous use of information and communication technology, even science and engineering students are required to learn programming and to take, at least, an introductory programming course (Malhotra & Anand, 2019). Furthermore, in light of Industry 4.0, programming education is becoming increasingly important (dos Santos, Vianna Jr, & Le Roux, 2018). Programming education deals with development of programming knowledge and skill. Programming knowledge involves understanding of programming concepts and learning a programming language. Programming skill is basically problem solving skill where programming knowledge is applied to create a program that solves a given problem. Traditionally, the instructional method for programming knowledge and skill development has been to get students to solve problems. Alternatively, programming instructors may use worked examples (Renkl, 2014). A worked example basically contains a problem specification and a worked-out solution. It has been found that example-based learning is more effective than problem solving

especially for students or novice problem solvers (Atkinson, Derry, Renkl, & Wortham, 2000; Renkl, 2014; Sweller & Cooper, 1985). However, in order for students to fully benefit from studying worked examples, they must actively process the example content (Renkl, 2017). They need to carefully read and cognitively process it. More specifically, they should make an effort to understand what programming concepts have been applied and to explain to themselves how the solution addresses the problem. A promising approach to foster self-explanation is to embed explanations in different sections of the solution in the form of textual labels (Atkinson, Catrambone, & Merrill, 2003; Catrambone, 1998). Recently, there has been research interest in the use of worked examples with labels in the programming domain (Margulieux & Catrambone, 2016; Morrison, Margulieux, & Guzdial, 2015). In a labelled worked example, statements in the solution program are grouped and labelled to explain how each group addresses different aspects of the problem. The labels embedded in the solution describe the design of the solution with respect to the problem requirements or subgoals. Consequently, they have been named subgoal labels (Catrambone, 1998).

Programming involves both problem analysis and solution generation skills. But, novice programmers or beginners tend to quickly jump to solution creation without adequately analysing the problem. Loksa and Ko (2016) found that very few of the participants in their study engaged in problem analysis. This lack of problem analysis skill is not limited to the programming domain. In a study among mathematics students, for instance, Schoenfeld (1992) found that students spent only a short time, at the beginning of their problem solving activity, reading the problem and the rest of the time exploring a solution, compared to an expert who spent half of the time first trying to understand a complex problem. Thus, novice programmers should be made aware of the importance of problem analysis in addition to solution generation. They should be supported to learn how to analyse problems as well as create solutions.

In light of this need, the current study proposed a new worked example design. The first objective for the design was to help create awareness in students that problem analysis should be the first phase in the programming process where problem requirements are identified. It must also serve to illustrate how problems may be analysed. So, instead of inserting subgoal labels in the solution program, the current study proposed to list the requirements or subgoals identified for the given problem, separate from the solution program. But, this means that the links between the subgoals and associated solution statements are lost. Hence, the second objective for the new worked example design was to find another way to show these connections so that students are able to see which part of the solution addresses those subgoals, which in turn, encourages them to self-explain the connections. The current study proposed the use of web technology to draw their attention to these connections in order to help students visualise them.

Consequently, the first research question for the current study was: (1) How can worked examples be designed and implemented for teaching and learning introductory programming at tertiary level to satisfy the two objectives stated above? The current study also sought to evaluate the effectiveness of worked examples based on the proposed design compared to worked examples with subgoal labels for learning programming. Hence, the second research question was: (2) Is there any difference in programming performance of students using worked examples based on the proposed design and worked examples with subgoal labels? It is hypothesised that worked examples based on the proposed design would be more effective for learning introductory programming.

THEORETICAL FRAMEWORK

The current study draws on research from three different areas: computer science education (or more specifically, programming education), example-based learning, and technology-supported learning which is a subset of the larger area of education research. The following subsections discuss the findings and issues from these areas that are relevant to the current study.

Programming Education

Teaching and learning programming involves not only knowledge of programming concepts but also of the programming process. The main stages in programming process model as applied by researchers in

studies on teaching and learning programming (e.g. Shi et al. (2019)) are to understand the problem by breaking it down into smaller problems, and to design, implement, test, and debug a solution. The process model specified by McCracken, Almstrum, Diaz, et al. (2001) prescribed these steps: extract relevant information about the problem and generate subproblems; decide on solution strategies or sub-solutions and integrate them into a complete solution, and; evaluate whether it solves the overall problem. These programming process models concur that the first and important step is to analyse or understand the problem. For this reason, it is important that students realise that problem analysis is a necessary first step in the programming process. Moreover, they need to be supported in learning how to analyse programming problems.

Example-Based Learning

Learning from worked examples has been proposed as an alternative to learning through problem solving (Renkl, 2014). Since students do not expend cognitive effort in generating the solution, learning from worked examples requires less cognitive load, thereby freeing up more cognitive resources for knowledge construction (Sweller, 1994). Learning from worked examples is especially useful in initial stages of problem solving skill acquisition when cognitive load is high because students do not yet have the necessary knowledge. Atkinson et al. (2000) reviewed numerous studies that provide evidence of the effectiveness of using worked examples compared to learning through conventional problem solving. More recent studies (e.g. (Cardellini, 2014; Van Gog, Kester, & Paas, 2011) also corroborate those earlier ones.

However, the design of worked examples must take into consideration the following two issues. Firstly, worked examples must be appropriately structured in order to facilitate knowledge construction (Atkinson et al., 2000; Moreno, 2006). Information should be presented in a form that assists students to construct and organise knowledge so that future retrieval of the knowledge matches the knowledge requirements of the tasks at hand (Driscoll, 2005). For the current study, the intended learning outcome of worked example study is construction of knowledge of common subgoals and their associated solutions to support subsequent programming problem solving tasks. Hence, the worked example should be structured in a manner that serves that purpose.

Secondly, for effective learning from worked examples, students must actively process the information presented. Renkl (2014) highlighted the principle of self-explanation as crucial for learning from worked examples. Students explain to themselves the rationale of the solution. Some ways to promote self-explanation are to insert labels explaining the steps in the solution, as in labelled worked examples (Catrambone, 1998) and to present two or more examples that can be compared and contrasted (Patitsas, Craig, & Easterbrook, 2013).

Technology-Supported Learning

An important factor for learning is engagement. Student engagement has been defined in different ways and different types of student engagement have been conceptualised (for example, behavioural, emotional, and cognitive (Schindler, Burkholder, Morad, & Marsh, 2017)). For the purpose of the current study, student engagement is taken to mean cognitive engagement which is defined as “the degree to which students invest in learning and expend mental effort to comprehend and master content” (Schindler et al., 2017, p.19). Different indicators of student engagement have been identified such as persistence in learning, academic achievement, and knowledge construction (Henrie, Halverson, & Graham, 2015; Schindler et al., 2017). The current study regards knowledge construction as the indicator of student engagement. Students must engage with the learning material in order for them to process its content and build their knowledge. The underlying assumption for the current study is that the more students are engaged during worked example study, the better will be the knowledge gained, which in turn, improves future programming performance.

Technology may be used to promote student engagement. The findings of (Dunn & Kennedy, 2019) suggested that engagement with technology-supported instruction was positively correlated with student achievement at university level. Schindler et al. (2017) reviewed the literature over a five-year period to

examine the influence of technology on student engagement. Their study focussed on students at tertiary education level. Their findings suggested that some types of technology may have positive influence on student engagement but that more research is still needed. They also recommended that, in instructional design practice, technologies should be selected carefully, taking into account the intended learning outcomes. Similarly, Spector (2013) stressed that learning goals must be considered when using technology for instruction and that research in technology-supported learning should examine how technology can generate engaging learning experiences.

An important aspect of student engagement is attention. Attention is a critical factor for learning to occur (Bester & Brand, 2013). Students must cognitively attend to the learning material presented to them. Their attention must be directed to the target that they should focus on (Bester & Brand, 2013). Research on technology-supported learning shows promise in the use of technology to capture student's attention. Bester and Brand (2013) examined the effect of technology on attention and achievement. They conducted an experiment with two groups: one was provided instruction supported by technology and the other without technology support. They adopted the definition of attention as "a system of cognitive control in which the vast amount of information processed by the cognitive system is reduced to a tolerable level" (Bester & Brand, 2013, p. 13). In the context of the current study, attention is seen as a cognitive process whereby a student focuses on a specific part of the worked example while ignoring others at any one time. This permits a tolerable level of cognitive processing of that part. The results of the study in (Bester & Brand, 2013) showed statistically significant differences between the groups for both achievement and attention. Their results suggested that use of technology for instruction has positive effect on learning achievement and attention. In addition, they showed that attention, concentration, and motivation were correlated to achievement, which implied that it is highly likely that achievement improves if technology is able to capture the attention of students during instruction. Although the study was conducted among school children (12-13 year olds) and the topic areas were diverse (geography, english, and mathematics), the notion that technology may be used to capture a student's attention, which in turn, may help improve a student's achievement, is plausible for university level students studying programming as well.

Reading and understanding worked examples designed for programming domain requires the student to be cognitively engaged with the content. The student must be motivated to read the solution program and to understand how each part of the program addresses different aspects of the problem. Technology-supported learning is envisaged to support student engagement in studying worked examples for the programming domain by drawing their attention to the various parts of the solution program and its relation to the problem.

DESIGN AND IMPLEMENTATION OF WORKED EXAMPLES FOR PROGRAMMING DOMAIN

This section introduces the design and implementation of worked examples for teaching and learning introductory programming at tertiary level to answer the first research question. The design is also targeted to satisfy the two objectives stated earlier. In line with a general design, a worked example has a problem specification and a worked-out solution. For the programming domain, the solution is a complete program that solves the given problem. Besides, in order for students to understand how the program works, the new design includes a sample run section. It shows the output produced by the program given some sample input data. To achieve the first objective of emphasising problem analysis, the new design introduces a problem analysis section which contains a list of subproblems (in other words, the product of problem analysis). Thus, each worked example is structured to have four sections: problem, analysis, solution, and sample runs. For the analysis section, since common subproblems exist for similar types of problems, subproblems are generalised into broad categories and presented at two levels: subproblem category and subproblem details. The subproblem details are specific to the problem at hand but the subproblem category is more general. Furthermore, to emphasise that subproblems are issues that students should consider when analysing a problem, a subproblem category is listed as a question and the subproblem detail becomes the answer to the question.

Web technology (HTML5, CSS3 and JavaScript) is used to implement the worked example design. Each

worked example is built as a web page containing four sections as shown in Figure 1. The web page is interactive in the sense that it responds to a user's action, specifically the user's mouse movements. The web page also supports highlighting which is implemented as change in background colour of different parts of the content whenever the user moves the mouse over a subproblem in the analysis section. These features of interactivity and highlighting are used for two purposes.

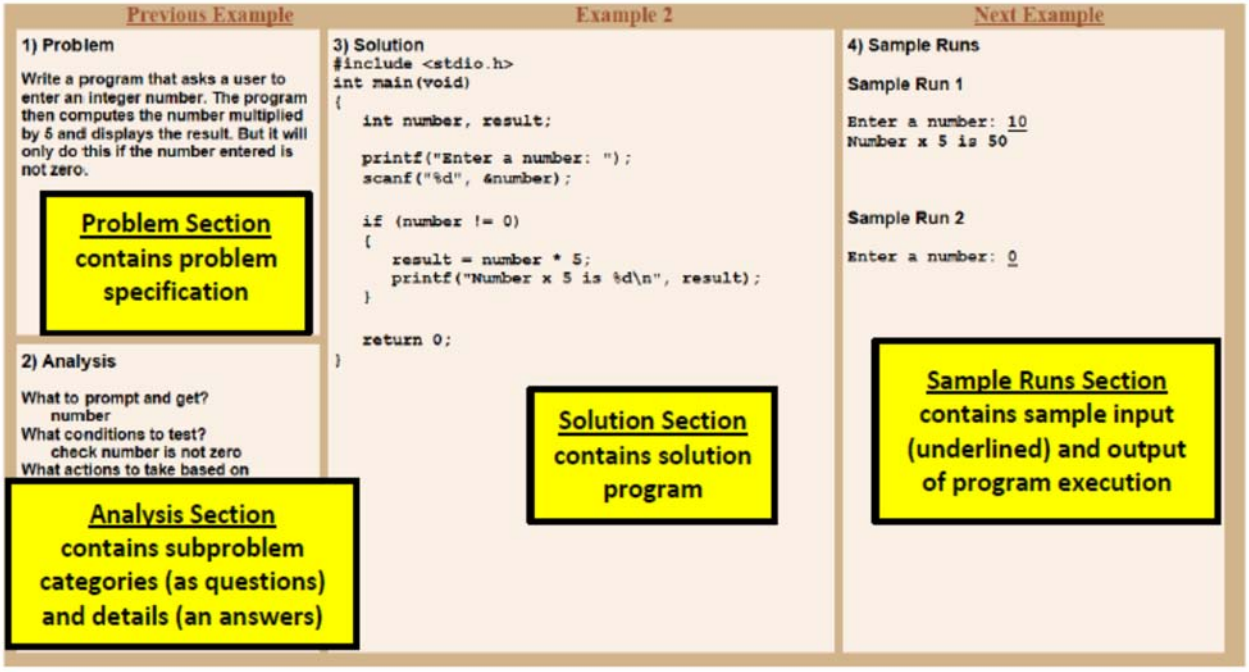

Figure 1. Example of a worked example with four sections

Firstly, interactivity and highlighting are used to achieve the second objective of the worked example design, which is to help students visualise the connection between a subproblem and the parts of the solution that fulfil that subproblem. Whenever the user moves the mouse over a subproblem, the elements in the solution program, that solves the subproblem, are highlighted together with the subproblem. Secondly, they are used to highlight the different elements in the problem specification associated with a subproblem. The intention was to help students visualise the connection and illustrate how subproblems are derived from the problem specification. Examples of these highlightings are shown in Figure 2, where the highlighted elements have a blue colour background.

Another type of highlighting is the blue-lined box around some statements in the program. The purpose of this highlighting is to enable students to visualise the boundary of a block or control structure (such as the selection control structure in Figure 2) so that they realise which statements are inside the control structure and which are outside its boundary. The aim was to give more clarification on the extent of a block structure.

Additionally, when worked examples are used for teaching and learning, they should be presented as a set (Renkl, 2014) of at least two. Typically, the examples chosen to form a set share similar subproblems, i.e. they are structurally or logically similar problems. As students study and compare the different examples in the set, they should recognise the structural similarities of the problems. They should then be able to generalise information from the examples and construct knowledge about the characteristics of different types of problem. To implement this feature, hyperlinks are added at the top of each worked example web page for navigation to the next and/or previous page, as shown in Figure 2, in order to allow students to move through the set.

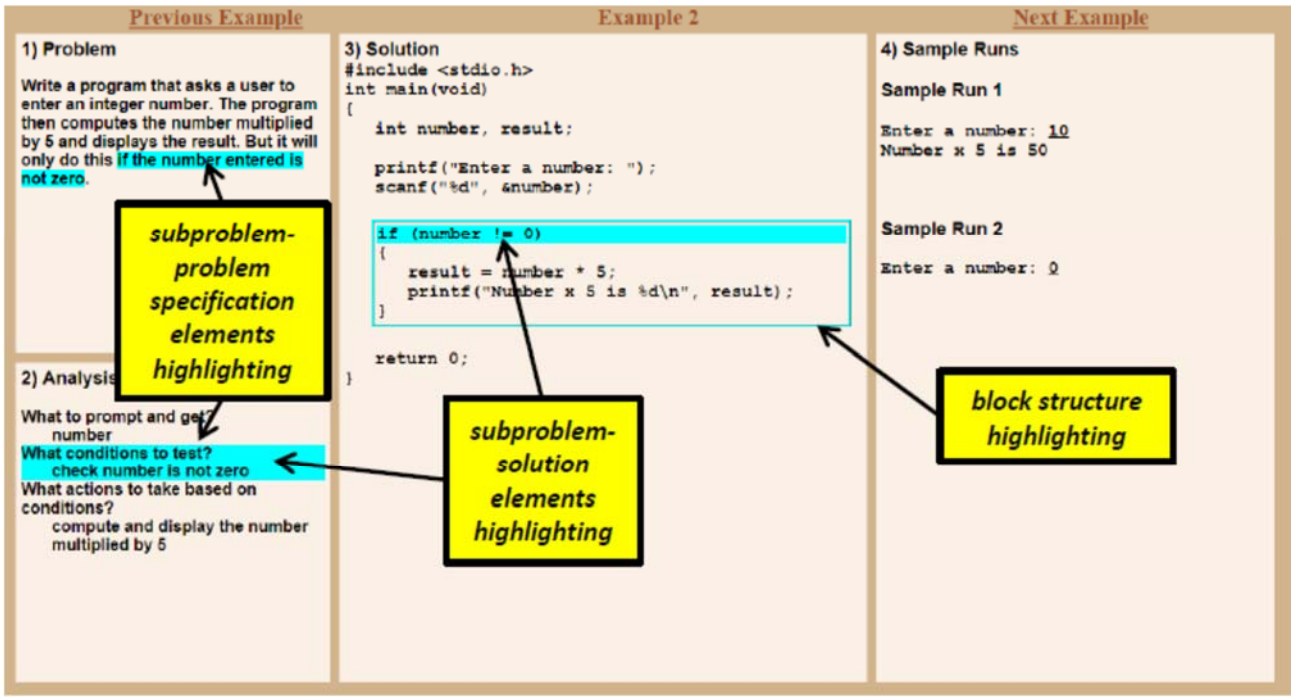

Figure 2. Example of a worked example with highlighted elements

EMPIRICAL STUDY

To evaluate the effectiveness of worked examples based on the proposed design compared to worked examples with subgoal labels for learning introductory programming, and answer the second research question, a pretest-posttest quasi-experiment with control and experimental groups was conducted in an introductory programming course at a university in Malaysia in the third trimester of 2017. The sampling technique used was purposive sampling (Etikan et al., 2016) where the criteria for participation were that the student must be enrolled in an introductory programming course and must be learning programming for the first time. The independent variable was the worked example design with two levels: worked example based on the proposed design (for experimental group) and labelled worked examples (for control group). The dependent variable was programming performance which was operationalized as a score achieved for programming problem solving questions given in the posttest. The pretest provided a way to determine the equivalence of the groups in terms of programming performance and knowledge prior to the learning activity. The pretests and posttests provided the means to collect quantitative data on programming performance which was analysed using statistical analysis.

The Course

The introductory programming course was one of the mandatory courses in a pre-degree (or foundation) programme leading to undergraduate degree programmes. The course syllabus was based on the procedural programming paradigm and the programming language used was C. The C programming language is commonly taught in programming courses for engineering programmes and there has been research interest in teaching and learning of such courses (e.g. (Altintas, 2016)). The introductory programming course was delivered through weekly two-hour lectures and two-hour practical classes in computer laboratories. During a practical class, students were expected to solve programming problems by applying concepts presented during lectures prior to the class. The experiment was conducted during three practical class sessions.

Participants

The participants were students from four intact practical classes of the course. Two of the classes were assigned to the control group and the remaining two to the experimental group. A majority (more than 95%)

of the participants were learning programming for the first time. Participants who had studied programming in prior courses were excluded in the data analysis. All participants gave consent for participation in the study. Ethical clearance was given by the ethics approval committee of the university. The participants were male and female students, aged between 18 and 19 years old, enrolled in a pre-degree programme leading to undergraduate engineering or science programmes.

Learning Materials

The worked examples for each experimental study session were created based on programming concepts, as listed in Table 1, which were part of the course syllabus and were commonly found in introductory programming textbooks. They were reviewed by three subject matter experts who had 15 or more years of experience teaching programming-related courses at tertiary level. The worked examples for both the control and experimental groups contained the same problems and solutions, and were arranged in the same order. They differed only in their designs.

Table 1. Concepts illustrated and tested (with maximum scores) for the 3 sessions

Pretest item			Worked example		Posttest item		
No.	Max. score	Concept tested	No.	Concept illustrated	No.	Max. score	Concept tested
Session 1							
1	7	if with else	1	if with else	1	9	nested if-else
			2	if without else	2	4	if without else
			3	nested if-else	3	7	if with else
					(Max. total score = 20)		
Session 2							
1	5	counter-controlled loop	1	counter-controlled loop	1	7	sentinel-controlled loop
			2	user-controlled loop	2	5	counter-controlled loop
			3	sentinel-controlled loop	3	9	user-controlled loop
					(Max. total score = 21)		
Session 3							
1	7	if nested in loop, and accumulator	1	if nested in loop	1	7	if nested in loop, and accumulator
			2	loop and accumulator			
					(Max. total score = 7)		

For the control group, each worked example had only three sections (problem, solution, and sample runs) and labels were inserted in the solution program as comments (the lines starting with //), as shown in Figure 3. The comments represented the subgoals which corresponded to the subproblems listed in the analysis section in the proposed design. (For the remainder of the paper, worked examples based on the proposed design in the experimental group are called visualised worked examples and worked examples used by the control group are called labelled worked examples.) Although the labelled worked examples could have been printed on paper, they were presented in web pages so that the medium of delivery for both groups was controlled. There was no interactivity or highlighting in labelled worked examples but navigation links were included at the top.

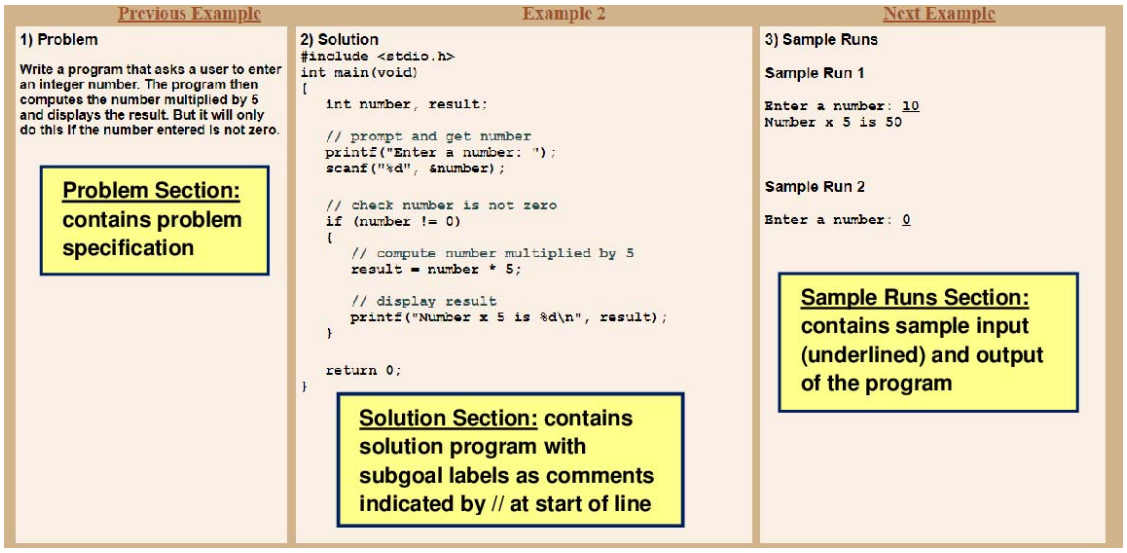

Figure 3. Example of a labelled worked example with labels in the solution.

Procedure

For each of the three class sessions, the stages of the experimental study were the same: pretest, independent learning activity (studying worked examples) and posttest. These were conducted first, after which the instructor conducted normal class activities. Posttests were conducted for each session, rather than only after the final session, so that the learning activity could be isolated as the main factor influencing the posttest scores and other factors, such as the instructor's teaching or discussions among students, after their class, could be controlled. Similarly, pretests were conducted for each session so that the equivalence of the two groups in terms of programing performance prior to the learning activity could be determined.

Participants were allowed to spend as much time as they needed for the pretest, independent learning activity, and posttest. However, an overall time limit (60 minutes for sessions 1 and 2 and 30 minutes for session 3) was set. Both groups studied worked examples and performed the pretests and posttests using a web browser. Participants were not permitted to refer to the worked examples when answering the pretest and posttest questions. Neither were they allowed to discuss with other participants or refer to any other learning materials. In other words, the tests were conducted in examination-like conditions. The pretest and posttest scores did not contribute to the grade for the course. In the week prior to the first study session, participants were introduced to the study, invited to participate, and completed demographic information and consent forms.

The pretests, learning activity, and posttests for all sessions were conducted through the use of a web browser. The first web page gave an introduction to the session with information on what to expect in the following web pages. The participant was required to enter their name which would be recorded in the web server. The next web page presented the pretest item where the given problem was displayed together with a text input area where the participant typed the answer program. After typing the answer, the participant submitted it and it was saved on the web server. The next web page contained information about the learning activity with instructions to study the worked examples carefully. Then, the web pages containing the worked examples for the learning activity were presented. After the participant had studied the worked examples, they proceeded to the remaining web pages which presented the posttest items on individual web pages one by one. For each posttest item, the participant's submitted answer was saved on the web server.

Measurement and Analysis

Programming performance of the participants was measured using programming assessment items in pretests and posttests. All items were problem solving questions which required participants to create a complete program as a solution to the problem. This reflected the knowledge and skill expected to be gained by the participants from studying the worked examples, which showed how problems were solved through application of different programming concepts. Because of the problem-solving nature of the test items, which required time for the participants complete, the number of items for the posttest was kept small. The posttest for sessions 1 and 2 were limited to 3 items since they were isomorphic to the 3 worked examples presented in those sessions, with the same structure as the worked example problems but different surface features. For session 3, only 1 item was used for the posttest because the solution involved the merging or composition of two programming concepts presented in the 2 worked examples for that session. The posttest items were given in the same sequence for both the control and experimental groups. As the performance on the pretest was expected to be low, the pretest was limited to only one item for all three sessions in order to sustain the motivation of the participants to complete the learning activity and posttest (Mulder, Lazonder, & De Jong, 2014). Table 1 shows the number of items, concepts tested, and maximum scores for the pretest and posttest items for each session. The pretest item for each session consisted of only one question which asked participants to write a program to solve a problem using the programming concept that was to be presented in the worked examples for that session. Scores were given for different statements in the answer program with a maximum score of 7, 5, and 7 for each of the three sessions respectively. For the posttest, there were three items for sessions 1 and 2, and one item for session 3. The posttest items were meant to assess the participants' programming performance after they had studied the worked examples for the sessions. For each posttest item, the participants had to write a program to solve a problem, just as for the pretest items, and scores were given for different statements in the program, as was done for the pretest items. The maximum total score for the posttest items was 20, 21, and 7 for each of the respective sessions.

The pretest and posttest items and rating protocol were reviewed by the three subject matter experts. For session 1, the pretest and posttest responses for two of the classes (one control and one experimental) were scored by two independent raters based on the rating protocol. This constituted responses from 49% of the total number of participants for session 1. The intraclass correlation coefficient (ICC) obtained was .93 which indicated that the interrater reliability for the 2 raters' scorings was high. The responses for remaining sessions were rated by one rater. Independent samples t-test was used to compare the experimental and control groups' pretest and postest score means. A p value (two-tailed) of less than .05 was considered statistically significant. In addition, frequency distributions of posttest scores across 4 different ranges (roughly equivalent to 4 quartiles) for the two groups were compared.

FINDINGS

Session 1

Data was collected from a total of 87 participants for this session. Demographic information for the experimental and control groups is shown in Table 2.

Table 2. Demographic information for session 1

Group	Female	Male	Engineering	Science	N
Experimental group	13	31	22	22	44
Control group	12	31	29	14	43

The independent samples t-test analysis to compare pretest performance of the experimental and control groups showed that there was no significant difference between the experimental group (M=1.95, SD=1.509) and control group (M=1.60, SD=1.81), as shown in Table 3. Hence, the two groups were considered comparable in programming ability and knowledge prior to the learning activity.

Table 3. The t-test analysis of the pretest score means for session 1

Group	N	Mean	Std. Dev.	t	df	p

Experimental group	44	1.95	1.509	0.98	85	0.33
Control group	43	1.60	1.81			

The t-test analysis to compare performance on posttest showed that participants in the experimental group had significantly better performance (M=13.93, SD=4.90) than those in the control group (M=11.26, SD=6.08), as shown in Table 4. The effect size (Cohen's d) was 0.5 which indicated a medium effect on learning for visualised worked examples compared to labelled worked examples.

Table 4. The t-test analysis of the posttest score means for session 1

Group	N	Mean	Std. Dev.	t	df	p
Experimental group	44	13.93	4.90	2.26	85	0.03
Control group	43	11.26	6.08			

The distribution of posttest scores across different ranges for both groups is graphically represented in Figure 4. It shows that 75.0% of participants in the experimental group obtained 50% or more of the maximum total score (i.e. scores of between 11 and 20) compared to 60.5% for the control group. These results suggest that visualised worked examples were more effective for learning than labelled worked examples.

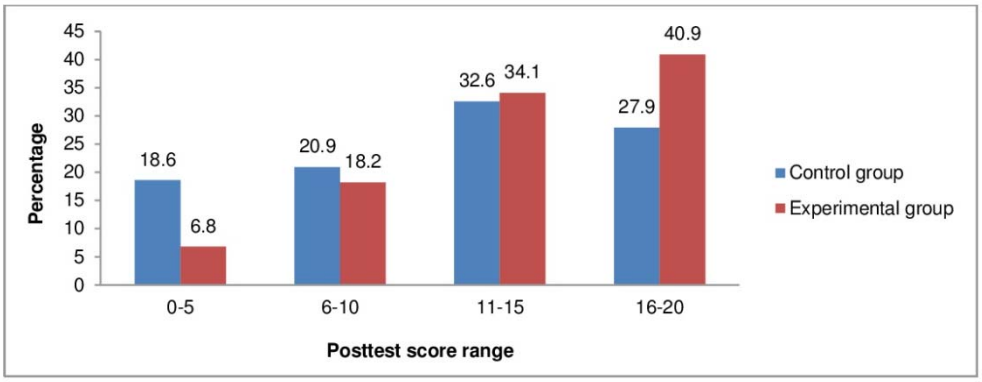

Figure 4. Distribution across posttest score ranges for the 2 groups for session 1.

Session 2

There were a total of 79 participants in session 2 (note: the total number of participants varied across sessions due to student absence from class). Demographic information for the experimental and control groups is shown in Table 5.

Table 5. Demographic information for session 2

Group	Female	Male	Engineering	Science	N
Experimental group	12	27	19	20	39
Control group	9	31	27	13	40

There was no significant difference in pretest performance between the experimental group (M=1.00, SD=1.61) and control group (M=1.00, SD=1.84) (Table 6). For the posttest, participants in the experimental group had marginally better performance (M=7.74, SD=5.97) than those in the control group (M=7.23, SD=6.50) but the difference was not statistically significant (Table 7).

Looking at the distribution of posttest scores (Figure 5), it is observed that about two-thirds of the participants in the experimental group (66.6%) and the control group (67.5%) obtained a score of 10 or less (i.e. less than 50% of the maximum score). This implies that the complexity of the test items (and correspondingly the worked examples) was high. However, the percentage of participants who performed poorly (in the lowest range of 0-5) in the experimental group (41.0%) was slightly smaller than for the control

group (45.0%). On the other hand, the percentage of participants who performed well (in the highest range of 16-21) in the experimental group (15.4%) was double that of the control group (7.5%). This suggests that the visualised worked examples were more helpful for learning than labelled worked examples.

Table 6. t-test analysis of the pretest score means of the two groups for session 2

Group	N	Mean	Std. Dev.	t	df	p
Experimental group	39	1.00	1.61	.00	77	1.00
Control group	40	1.00	1.84			

Table 7. t-test analysis of the mean posttest score means of the two groups for session 2

Group	N	Mean	Std. Dev.	t	df	p
Experimental group	39	7.74	5.97	.37	77	0.71
Control group	40	7.23	6.50			

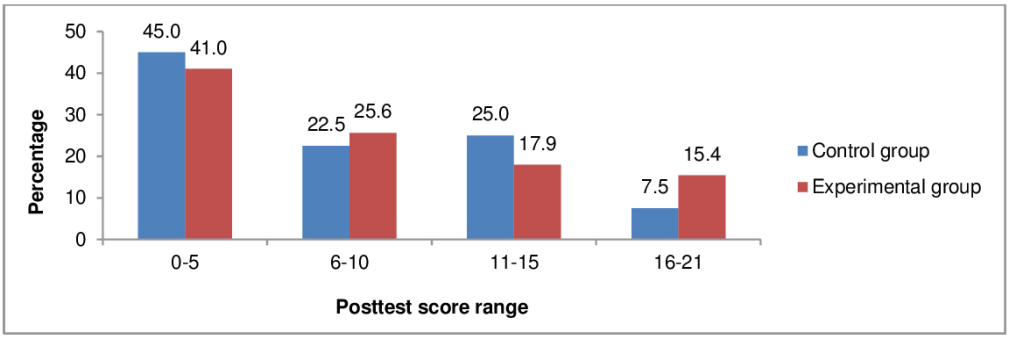

Figure 5. Distribution across posttest score ranges for the 2 groups for session 2

Session 3

A total of 78 students participated in the study for session 3. Demographic information for the experimental and control groups is shown in Table 8. There was no significant difference in pretest performance for the experimental (M=1.67, SD=2.39) and control groups (M=1.54, SD=2.211) (Table 9). As was the case for session2, participants in the experimental group had marginally better performance (M=3.79, SD=2.05) than those in the control group (M=3.59, SD=2.09) for the posttest and the difference was not significant (Table 10).

From the distribution of posttest scores (Figure 6), it is observed that a higher percentage of participants in the experimental group (56.4%) obtained scores of 50% or more of the maximum score (i.e. between 4-7) compared to the control group (48.7%). This suggests that visualised worked examples were more beneficial for learning than labelled worked examples.

Table 8. Demographic information for session 3

Group	Female	Male	Engineering	Science	N
Experimental group	13	26	18	21	39
Control group	10	29	25	14	39

Table 9. The t-test analysis of the pretest score means for session 3

Group	N	Mean	Std. Dev.	t	df	p
Experimental group	39	1.67	2.39	0.26	76	0.81

Control group	39	1.54	2.21			

Table 10. The t--test analysis of the posttest score means for session 3

Group	N	Mean	Std. Dev.	t	df	p
Experimental group	39	3.79	2.05	.44	76	0.66
Control group	39	3.59	2.09			

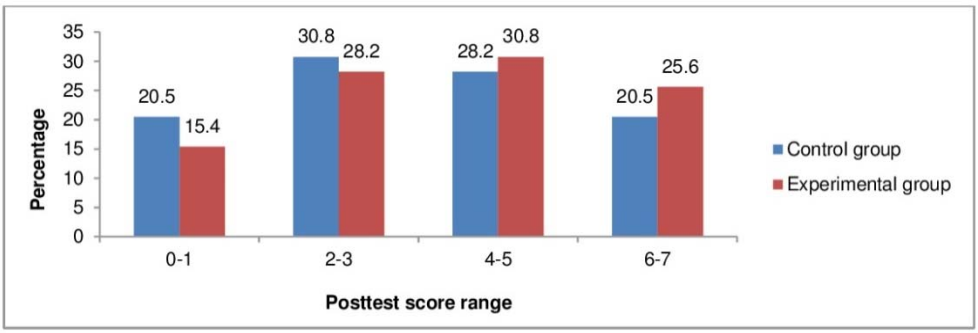

Figure 6. Distribution across posttest score ranges for the 2 groups for session 3

DISCUSSION

The empirical study compared the performance of learning from visualised worked examples and labelled worked examples to answer the second research question. With programming performance as an indicator of learning gains, in general, the results suggested that visualised worked examples were more effective than labelled worked examples for learning introductory programming, as was hypothesised. For session 1, the difference in programming performance after worked examples learning was statistically significant and there was a medium effect. This suggested that visualised worked examples provided much better learning gains from example-based learning than labelled worked examples. The interactivity and highlighting that emphasised specific elements in the problem analysis, problem specification, and solution sections, may have guided participants' attention to the relevant parts of the worked example to focus on and relate together. This could have encouraged better processing of learning content and led to better learning, and subsequently, better performance. This finding is consistent with previous studies on visual highlighting (Jamet & Fernandez, 2016; Lin, Atkinson, Savenye, & Nelson, 2016) which showed positive effects on learning.

For sessions 2 and 3, although visualised worked examples led to higher programming performance for learning from worked examples, the difference was marginal, probably because of the complexity of the content and the programming assessment. If the worked examples were too complex for the students to understand, it could explain the low learning achievement for both experimental and control groups. The complexity could be caused by high element interactivity (Van Merriënboer & Sweller, 2005) which refers to the degree of interrelationships among the elements in learning materials that students have to attend to simultaneously. The worked examples used in session 2 contained more components that participants had to focus on and relate together compared to those used in session 1. Furthermore, the programming assessment in session 3 was more complex than that for session 1 because participants had to merge two programming concepts presented in the two worked examples for that session and compose the solution. These factors probably limited the learning, and therefore, the performance.

Some threats to validity in the current study are noted as follows. Since intact classes were used, the participants were not randomly assigned to the two groups. However, the pretests provided to a way to evaluate the equivalence of the groups. There were no significant differences between the groups in the

pretests for all three sessions. The four classes were taught by three different instructors, which might have led to instructor effect. One of the instructors handled two of the classes. To balance out any instructor effect, one of the two classes taught by the same instructor was assigned to the experimental group while the other to the control group. However, instructor effect would be minimal since the learning activity involved independent learning without any instructor assistance. Also, the posttest was conducted immediately after the learning activity, before the instructors' normal class activities. However, there is a possibility that the instructors' activities could affect the subsequent study session. But, the same lecture notes and practical sheets were used for all classes. Again, the pretests provided a way to determine any influence of instructors, and it was found that the control and experimental groups had comparable programming performance prior to the learning activity for all three sessions.

CONCLUSION

This paper proposed a new design for worked examples to be used for teaching and learning introductory programming among tertiary education students. The design intentions were to encourage students to attentively examine the learning content, and hereby improve their programming knowledge, as well as to assist in development of their problem analysis and solution generation skills in programming. The proposed worked example design was implemented and evaluated in an empirical study to compare its effectiveness for learning introductory programming with subgoal labelled worked example design. The study results suggested that the proposed design was more effective for example-based learning for some topics and equally as effective for others. Just as subgoal labelled worked examples are theorised to support better learning from examples because they encouraged students to self-explain the example content, the empirical results of the current study suggested that the proposed design may provide as good or even better support.

The current study contributed to empirical research on the use of worked examples for learning introductory programming at tertiary level. Further research is needed to improve the proposed worked example design and implementation to cope with complexity of learning content. Furthermore, studies similar to the current should be conducted at other tertiary education institutions where introductory programming is taught. Based on the potential usefulness of the proposed design, programming instructors at tertiary education institutions who wish to adopt example-based learning could utilise the proposed design for worked examples in order to improve the effectiveness of learning from worked examples.

REFERENCES

Altintas, T., Gunes, A., & Sayan, H. (2016). A peer-assisted learning experience in computer programming language learning and developing computer programming skills. *Innovations in Education and Teaching International*, *53*(3), 329-337. doi: 10.1080/14703297.2014.993418.

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. doi:10.2307/1170661.

Atkinson, R. K., Catrambone, R., & Merrill, M. M. (2003). Aiding transfer in statistics: examining the use of conceptually oriented equations and elaborations during subgoal learning. *Journal of Educational Psychology, 95*(4), 762-773. doi: 10.1037/0022-0663.95.4.762.

Bester, G., & Brand, L. (2013). The effect of technology on learner attention and achievement in the classroom. *South African Journal of Education*, 33(2), pp. 1-15.

Cardellini, L. (2014). Problem solving: how can we help students overcome cognitive difficulties. *Journal of Technology and Science Education, 4*(4), 237-249. doi: 10.3926/jotse.121.

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General*, *127*(4), 355-376. doi:10.1037/0096-3445.127.4.355.

dos Santos, M. T., Vianna Jr, A. S., & Le Roux, G. A. (2018). Programming skills in the industry 4.0: are chemical engineering students able to face new problems?. *Education for Chemical Engineers*, *22*, 69-76.

Driscoll, M. P. (2005). Psychology of learning for instruction, (3rd ed.) Boston: Pearson.

Dunn, T. J., & Kennedy, M. (2019). Technology enhanced learning in higher education; motivations, engagement and academic achievement. *Computers & Education*, *137*, 104-113.

Etikan, I., Musa, S. A., & Alkassim, R. S. (2016). Comparison of convenience sampling and purposive sampling. *American Journal of Theoretical and Applied Statistics*, *5*(1), 1-4. doi: 10.11648/j.ajtas.20160501.11

Henrie, C. R., Halverson, L. R., & Graham, C. R. (2015). Measuring student engagement in technology-mediated learning: A review. *Computers & Education*, *90*, 36-53.

Jamet, E., & Fernandez, J. (2016). Enhancing interactive tutorial effectiveness through visual cueing. *Educational Technology Research and Development*, *64*(4), 631-641. doi:10.1007/s11423-016-9437-6.

Lin, L., Atkinson, R. K., Savenye, W. C., & Nelson, B. C. (2014). Effects of visual cues and self-explanation prompts: empirical evidence in a multimedia environment. *Interactive Learning Environments*, *24*(4), 799-813. doi:10.1080/10494820.2014.924531.

Loksa, D., & Ko, A. J. (2016, August). The role of self-regulation in programming problem solving process and success. *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 83-91). ACM.

Malhotra, V. M., & Anand, A. (2019). Teaching a university-wide programming laboratory: managing a C programming laboratory for a large class with diverse interests. *Proceedings of the Twenty-First Australasian Computing Education Conference on - ACE '19*. doi:10.1145/3286960.3286961.

Margulieux, L. E., & Catrambone, R. (2016). Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction*, *42*, 58-71. doi:10.1016/j.learninstruc.2015.12.002.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working group reports from ITiCSE on Innovation and Technology in Computer Science Education - ITiCSE-WGR '01*. doi:10.1145/572134.572137.

Moreno, R. (2006). When worked examples don't work: Is cognitive load theory at an Impasse? *Learning and Instruction*, *16*(2), 170-181. doi:10.1016/j.learninstruc.2006.02.006.

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, context, and worked examples in learning computing problem solving. *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*. doi:10.1145/2787622.2787733.

Mulder, Y. G., Lazonder, A. W., & De Jong, T. (2014). Using heuristic worked examples to promote inquiry-based learning. *Learning and Instruction*, *29*, 56-64. doi:10.1016/j.learninstruc.2013.08.001.

Patitsas, E., Craig, M. & Easterbrook, S. (2013). Comparing and contrasting different algorithms leads to increased student learning. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* , San Diego, 145-152. doi: 10.1145/2493394.2493409.

Renkl, A. (2014). Toward an instructionally oriented theory of example-based learning. *Cognitive Science*, *38*(1), 1-37. doi:10.1111/cogs.12086.

Renkl, A. (2017). Learning from worked-examples in mathematics: students relate procedures to principles. *ZDM*, *49*(4), 571-584. doi:10.1007/s11858-017-0859-3.

Schindler, L. A., Burkholder, G. J., Morad, O. A., & Marsh, C. (2017). Computer-based technology and student engagement: a critical review of the literature. *International Journal of Educational Technology in Higher Education*, *14*(1), 25.

Schoenfeld, A. H. (1992). Learning to think mathematically: problem solving, metacognition, and sense making in mathematics. D. Grouws (Ed.) *Handbook for Research on Mathematics Teaching and Learning*. NY: Macmillan, 334-370.

Shi, J., Sha, A., Hedman, G., & Rourke, E. O. (2019). Pyrus: designing a collaborative programming game to support problem-solving behaviors. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow, Paper No. 656. doi: 10.1145/3290605.3300886.

Spector, J. M. (2013). Trends and Research Issues in Educational Technology. *Malaysian Online Journal of Educational Technology*, *1*(3), 1-9.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, *4*(4), 295-312. doi:10.1016/0959-4752(94)90003-5.

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. doi:10.1207/s1532690xci0201_3.

Van Gog, T., Kester, L., & Paas, F. (2011). Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, *36*(3), 212-218. doi:10.1016/j.cedpsych.2010.10.004.

Van Merriënboer, J. J., & Sweller, J. (2005). Cognitive load theory and complex learning: recent developments and future directions. *Educational Psychology Review*, *17*(2), 147-177. doi:10.1007/s10648-005-3951-0.