International Journal of Pure and Applied Mathematics
Volume 118 No. 7 2018, 77-83
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

Scaffolding Model for Efficient Programming Learning Based on Cognitive Load Theory

Syahanim Mohd Salleh, Zarina Shukur and Hairulliza Mohamad Judi
Faculty of Information Sciences and Technology,
Universiti Kebangsaan Malaysia, MALAYSIA
{syahanim,zarinashukur,hmj}@ukm.edu.my

***Abstract*—Programming learning for beginners requires tremendous amount of exposure to understand the logic in each programming solution using the basic concepts despite the overwhelming syntax it might carries. Learning programming through examples with careful walkthrough builds learners' confidence to embark with problems of any designs, avoids frustration due to syntax error and unintentional bugs. Scaffolding involves meta-programming approach of building software applications using supported materials that provides some inspiration of how the program could be developed. This research identifies important attributes in programming and proposes a scaffold model to enhance programming learning efficiency especially among novice programmers. The study applies cognitive load theory by providing users with two types of instructional design as learning support to reduce mental effort applied in the working memory i.e. worked-example and goal free programming problem solutions. The model is expected to help instructors in systematically organizing programming materials for any language or programming environment for efficient programming learning.**

Keywords*—programming, scaffolding, learning support, cognitive load, worked-example

I. Introduction

Computer programming entails writing instructions using an artificial language to be translated into machine language and then executed by a computer to solve various problems. Programming courses are offered at university level to introduce students with various programming concepts to be applied in solving problems from a broad range of fields. Programming teaching and learning is usually determined by programming concepts. To relate programming concepts with real life problem solving, programming tasks often use typical problems such as in determining grades from the final scores of statistics course. The use of common problems in trouble shooting facilitates students to understand the real problem and thus design the program execution.

There are certain programming concepts that need to be associated with a relevant problem for indicating its intended use. For example, the calculation of inflation rate and problem simulation in banks aim to explain certain concepts regarding choice and rotation. Due to very limited relevant knowledge and experience, novice programmers encounter difficulties to perform a given task (as in Fig. 1). Apart from describing programming concepts, course instructor should also explain relevant information whenever un-common problems are raised. These information may support students with the required skills and techniques to solve the problem.

Conventional programming instruction emphasizes theoretical programming concepts and associates them with word syntax and simple program examples [1]. The implementation of this teaching approach does not include explanation of related tasks to support students' understanding and to complete the given tasks. In addition, teaching materials such as lecture notes and slides do not provide any similar examples and problem solutions to the given assignment. Therefore, students are fully dependent on his / her own knowledge to construct a solution. Students' difficulty in understanding the basic problem increased cognitive burden especially to novice troubleshooters which leads to failure to solve the task [2].

The problems posed by the current scenario in an environment of problem solving in programming learning show the need for scaffolding to assist instructors in developing better problem statement in their assignments and for providing a support for students building their program [3]. Hence, to provide the support needed by these students, especially in programming learning, specific guidance should also be offered as a compilation of all required programming concepts, tasks, assignment statement, as a structured and organized material. This guide should include various levels of programming that is not influenced by any language or programming environment so that it is easily applied by any programming course by any institution. Thus, this research identifies important attributes in programming and proposes a scaffold model to enhance programming learning efficiency especially among novice programmers.

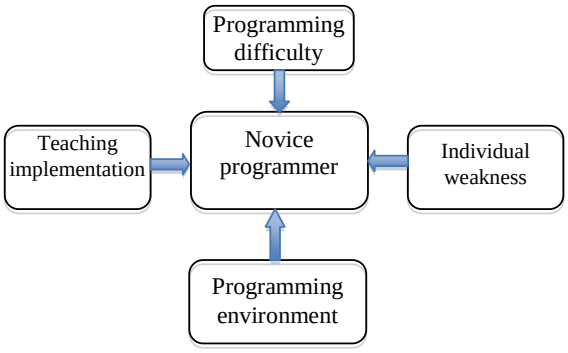

Fig. 1. Difficulties among novice programmer

II. Programming Learning Using Cognitive Load Theory

Cognitive load theory (CLT) was initiated by John Sweller [4] in the late 1980s based on a problem-solving study. In the area of cognitive psychology, cognitive loads refer to the amount of information processed by individual work memories. Since human memory can only process very limited information, a strategy that reduces cognitive loads and optimizes limited information is required in order to support the problem-solving process. Thus, cognitive load theory suggests changes and guidelines to the teaching design so that cognitive loads are reduced especially in the implementation of problem solving and exercise [4].

The three forms of guidelines recommended by CLT are (i) preventing excessive cognitive burden, (ii) reducing unrelated external cognitive burden in learning and (iii) increasing cognitive loads within the limitations of existing mental process, especially that are directly related to learning. Teaching design strategies and activities based on cognitive theories such as breaking information to several small segments and using goal-free problems can improve the ability of novices to understand the concepts and theories to support the problem solving process [5]. The study in CLT suggests that the reduction of cognitive loads of students by manipulating external cognitive loads through several methods involving structural changes, such as using a simple delivery method and applying the support of instructional exercises to be aligned with students' acceptance and cognitive ability [6].

Four recommended strategies based on CLT for learning and problem solving in programming are (i) completing solution, (ii) modular learning, (iii) work-example and (iv) goal-free problems. The discussion of each of these strategies involves the implementation of strategies, elements or basic attributes for implementing strategies, the effect of strategy implementation and the areas of learning that have been implementing strategies.

Completing Solution involves producing answers to certain parts of solution with other parts already being solved. The strategy stimulates students ability to modify or expand the program [7]. The implementation of this strategy allows students to complete the solution with reduced support gradually acting as tutors in student learning [8]. Modular learning strategy helps students minimize the intrinsic cognitive burden by gradually resolving problems from simple to complex surroundings [9]. Implementation of the strategy can guide novice cognitive skills to have a positive impact on their cognitive patterns and tasks [10]. Modular learning encourages problem with procedural solutions to be broken down into meaningful smaller units for better delivery and content to improve learning [11].

Task- example involves providing problem solution, where every step is fully stated and explicitly pointed out [12]. The strategy is used in teaching, where teachers provide work-related problems for students to learn. The integration of relevant and related information reduces the tedious searching process, thus reducing the burden on working memory. One of the alternatives to conventional problem solving is to use a goal-free problem approach. In this strategy, problem statements do not specify the objectives or tasks to be addressed. Goal-free problems can prevent students from finding the difference between the current state of the problem and the ultimate goal as the goal is not stated. In the process of solving goal-free questions, students are given the freedom to consider any problem situation and to identify possible solutions. This situation will then lead to a new situation or problem, where the student will deal with the next step of the solution.

The use of goal-free question in learning helps students to focus on current situations and solutions, which results in a reduction in student workforce burden. The goal-free strategy can reduce cognitive burden as students focus on the initial state instead of using means-ends [13] Since many computer programming is goal-oriented, the learning approach using question-free questions will seem difficult to implement, but it has significant and feasible value for intermediate level learning [6].

Due to its close relevancy to programming learning, two strategies, i.e. task-example and goal free approach will be implemented in the study. Despite their practical application in programming learning, these approaches have not yet discussed in detailed regarding their implementation in the domain based on CLT.

Based on the literature review, comparison analysis has been made to examine the implementation of the recommended CLT strategies to address students problem in programming course. The implementation of strategy in programming classes is specific for programming language, level and environments, as supported by specific tools. Table 1 provides the comparison. Majority of CLT strategies are related to specific training purposes and programming tasks although no clear explanation is available regarding how this strategy is justified in teaching sessions, especially to help students understand the problems and accomplish the given tasks.

.

TABLE I. COGNITIVE LOAD THEORY IN PROGRAMMING

Research	Teaching aid strategy	Programming Language	Level	Programming Tools
[14]	Completing solution	Visual BASIC	Introductory programming	CORT (Code Restructuring Tool)
[11]	Modular learning, Task-example	Un-named	Introductory programming	Problem solving tool HYPERCOMB
[7]	Completing solution	BASIC	Introductory programming	CLIPS, using MS-Windows.
[8]	Task-example	Un-named	Problem solution	Cognitive Tutorial for geometrical domain
[15]	Task-example, Modular learning, Completing solution	Un-named	Introductory programming	LECSES
[16]	Task-example	C#	Introductory programming	Visual Studio
[17]	Completing solution	Un-named	Introductory programming	Looking Glass
[10]	Task-example, scaffold	Un-named	Introductory object oriented programming	Not applicable
[18]	Completing solution	Lisp	Introductory programming	IPE
[19]	Modular learning	Any	Various programming level	Not applicable
[20]	Structured programming	Un-named	Introductory programming	Lego Mindstorms robots

III. DESIGN AND DEVELOPMENT

The design and development of proposed scaffold model to enhance programming learning efficiency especially among novice programmers is explored in detail in this section. The proposed model is aimed to assist instructors in developing better problem statement in their assignments and for providing a support for students building their program. The process flow to identify important attributes in programming is presented in Fig. 2, while the research flow for model development is illustrated as in Table II.

These figures represent three main activities in this study ie (1) identifying component attributes for two relevant CLT strategies, (2) designing scaffolding for the strategy and (3) modeling scaffolding for programming learning.

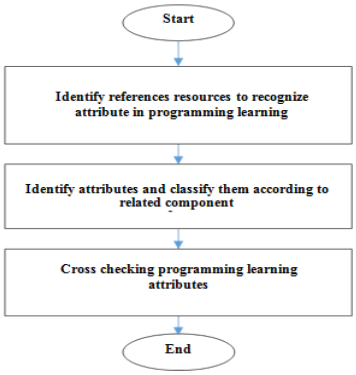

Fig. 2. Flow chart of identifying programming attributes

The learning model will consists of two problem classes i.e., (1) objective-oriented problem that will employ a task-example strategy to reduce novice cognitive burden and (2) goal-free problem that emphasize freedom to consider any problem situation and to identify possible solutions that has been magnified in cognitive load theory.

TABLE II. DEVELOPING SCAFFOLDING MODEL

Identify programming component attribute	
Data collection Literature review, Programming practice guideline, programming competition instruction and questions	
Worked-example strategy 1. Question statement component • Topic • Background • Formula • Figure • Task • Input and output 2. Support component • Question statement • Algorithm • Full program	Goal free strategy 1. Question statement component • Title • Background • Formula • Task • Exhibit 2. Support component (solution booster question) • task • relevant data • process • interface
Attribute and component list	
⇩	
Scaffold model	
Validation by expert	

IV. SCAFFOLDING MODEL

Implementation of the scaffold model in this study focuses on programming problem solving. The scaffolding implementation component (as shown in Table III and Fig.3) for this study consists of two main divisions namely Assignment and Scaffolding.

Assignment component concentrates on problem solving task. Assignment contains a statement according to the category of problem being represented. Statement of objective-oriented problems includes detailed information and expressing the tasks that need to be done clearly. While the question statement for the mathematical problem involves the disclosure of information pertaining to the background of the problem and the tasks that need to be implemented in general.

Scaffolding component is used to support problem solving. This study suggests different scaffolding based on the problem category. For the goal-oriented category, the model uses example-work as the main scaffolding as well as scaffolding implementation to support the implementation of task-example in teaching. In the case of goal-free strategy, this study adapts the case-study method involving scaffolding in the form of questions to trigger ideas for problem solving.

TABLE III. PROGRAMMING ATTRIBUTE AS SCAFFOLD COMPONENT

Scaffold design	
Task-example problem	**Goal-free problem**
1. Assignment: Question statement • *General info*: topic, category, level, C++ element, additional function • *Background info*: Scenario, additional info (formula, graph, table, exhibit attachment) • *Task* (specific) • *Input*: Boundary and examples • *Output*: Boundary and examples **2. Scaffold** ***A. Worked-example*** Worked-example complexity I. Question statement • Question attributes II. Solution • Algorithm • Programme basic structure • Full programme ***B. Support*** [1] Problem concept [Lecture] [2] Programming concept [Lecture] [3] Solution description [Tutorial] [4] Programming development [Lab]	**1. Assignment: Question statement** • *General info*: topic • *Background info*: Scenario, additional info (formula, graph, table, exhibit attachment) • *Task* (general) **2. Scaffold** Idea booster questions i. Problem surroundings ii. Process and task iii. Relevant data iv. Output of solution v. Interface

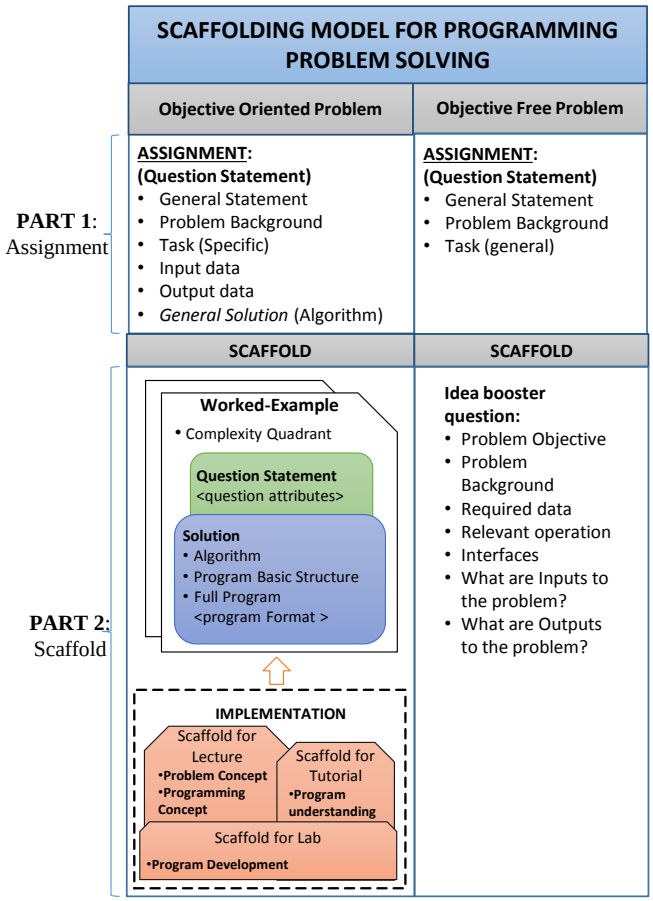

Fig. 3. Scaffolding model

V. Discussion And Conclusions

Scaffolding is among the proposed solutions for current scenario and issues in programming learning. This study proposes scaffolding model as a guide to assist instructors in developing better problem statement in their assignments and for providing a support for students building their program [3]. There were common difficulties faced by novice programmers that led to their nature to be of limited knowledge, skills and experience in problem-solving. Consequently, their background has led them to use a means-end solution method that has proven to increase cognitive burden [13]. For the purpose of reducing cognitive burden on student problem solving, the study referred to cognitive load theory (CLT) [4] and decided to employ task-example and goal-free strategies to be used in the scaffold model.

Cognitive loads are the main issues in programming learning and problem solving. In order to avoid novices using a mean-ends solution that increases cognitive loads in problem solving, cognitive load theory demonstrates task-example-solving strategies and goal-free questions. Although this strategy has been adopted in previous research, the detailed implementation of these two strategies in programming teaching and learning has not discussed. Hence, the analysis of four programming sources of reference is conducted to identify the key attributes that support programming teaching and learning such as problem backgrounds, task statements, input lists, output formats, relevant formulas and illustrations of the diagram for goal-oriented categories. For goal-oriented problems, the attributes involved are the trigger questions and the details of the problem background.

The study on two recommended CLT strategies identifies three key components in task-example strategy i.e. questions, solutions and implementation support. The main component in goal-free strategy is the statement of assignment questions and teaching notes that comprise of questions to trigger solution ideas. The two CLT recommendations are used in the different problem categories i.e. task-example recommendations for objective-oriented problem categories while goal-free question recommendations for objective-free problem categories.

The study has determined programming attributes and has designed scaffolding model for programming learning and problem solving using two recommended CLT strategies. The process of identifying attributes through references using four programming sources, namely programming books, programming competition questions, past programming practice statements and teaching case books. The results of this analysis identified a number of 12 attributes of the questions and 7 attributes of support for the goal oriented problem category. While 5 attributes for the question and 2 support attributes for the implementation of objective-free problem category. The list of attributed attributes is mapped to the components of the relevant recommendations.

To ensure the problems and solutions used by the task-example work can guide students in solving task problems, the relationship between assignments and work examples using four quadrant classes is determined by the problem background and the complexity of the problem. In addition to implementing scaffolding, the model also suggest the relationship between tasks and scaffolding involved in teaching process.

As a conclusion, this study has proposed scaffolding model as a guide to assist instructors in developing better problem statement in their assignments and for providing a support for students building their program. It is expected that the scaffolding model be executed among programming learning faculties within local and abroad higher institutions. The model may assist instructors in regulatory teaching and tackling cognitive loads issues in problem solving. Widespread use of scaffolding model in regulatory teaching can provide feedback for models improvement in future.

References

[1] A. Eckerdal, "Novice Programming Students' Learning of Concepts and Practise," Upssala University Thesis, 2009.

[2] T. de Jong, "Cognitive load theory, educational research, and instructional design: some food for thought," Instr Sci, vol. 38, pp. 105–134, 2010.

[3] M. Ueno and Y. Miyazawa, "IRT-based adaptive hints to scaffold learning in programming," IEEE Trans. Learn. Technol., vol. 14, no. 8, 2017.

[4] J. Sweller, "Cognitive load during problem solving: Effects on learning," Cogn. Sci., vol. 12, no. 2, pp. 257–285, 1988.

[5] S. Kalyuga, A. Renkl, and F. Paas, "Facilitating Flexible Problem Solving : A Cognitive Load Perspective," Educ Psychol Rev, vol. 22, pp. 175–186, 2010.

[6] D. Shaffer, D. Wendy, and J. Tuovinen, "Applying Cognitive Load Theory to Computer Science Education," in Applying Cognitive Load Theory to Computer Science Education, 2003.

[7] K. Chang, B. Chiao, S. Chen, and R. Hsiao, "A Programming Learning System for Beginners— A Completion Strategy Approach," in IEEE Transactions On Education, 2000, vol. 43, no. 2, pp. 211–220.

[8] R. J. C. M. Salden, K. R. Koedinger, A. Renkl, V. Aleven, and B. M. McLaren, "Accounting for Beneficial Effects of Worked Examples in Tutored Problem Solving," Educ. Psychol. Rev., vol. 22, no. 4, pp. 379–392, Sep. 2010.

[9] J. Sweller, "Element Interactivity and Intrinsic, Extraneous, and Germane Cognitive Load," Educ. Psychol. Rev., vol. 22, no. 2, pp. 123–138, 2010.

[10] M. E. Caspersen and J. Bennedsen, "Instructional Design of a Programming Course - A Learning Theoretic Approach," in Proceedings of the third international workshop on Computing education research: ICER 2007, 2007, pp. 111–122.

[11] P. Gerjets, K. Scheiter, and R. Catrambone, "Can learning from molar and modular worked examples be enhanced by providing instructional explanations and prompting self-explanations?," in Learning and Instruction, 2006, vol. 16, no. 2 SPEC. ISS., pp. 104–121.

[12] B. R. E. Clark, P. A. Kirschner, and J. Sweller, "Putting Students on the Path to Learning The Case for Fully Guided Instruction," Am. Educ., pp. 6–11, 2012.

[13] K. S. Sweller J., Ayres P., "The Goal-Free Effect. In: Cognitive Load Theory. Explorations in the Learning Sciences," Instr. Syst. Perform. Technol., vol. 1, pp. 89–98, 2011.

[14] S. Garner, "Learning To Program Using Part-Complete Solutions," 2000.

[15] S. S. Abdul-Rahman and B. Du Boulay, "Learning programming via worked-examples: Relation of learning styles to cognitive load," Comput. Human Behav., vol. 30, pp. 286–298, 2014.

[16] C. Vieira, J. Yan, and A. J. Magana, "Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design," J. Comput. Sci. Educ., vol. 6, no. 1, pp. 2–15, 2015.

[17] K. J. Harms, "Applying cognitive load theory to generate effective programming tutorials," in 2013 IEEE Symposium on Visual Languages and Human Centric Computing, 2013, pp. 179–180.
[18] P. H. Feiler and R. Medina-Mora, "An Incremental Programming Environment," in Proceedings of the 5th International Conference on Software Engineering, 1981, no. 5, pp. 44–53.
[19] W. Cazzola and D. M. Olivares, "Gradually learning programming supported by a growable programming language," IEEE Trans. Emerg. Top. Comput., vol. 4, no. 3, pp. 404–415, 2016.
[20] R. Mason and G. Cooper, "Mindstorms robots and the application of cognitive load theory in introductory programming," Comput. Sci. Educ., vol. 23, no. 4, pp. 296–314, 2013.