Aligning and Assessing Teaching Approaches With SOLO Taxonomy in a Computer Programming Course

Sohail Iqbal Malik, Buraimi University College, Al Buraimi, Oman
https://orcid.org/0000-0002-2737-9255
Ragad M. Tawafak, Buraimi University College, Al Buraimi, Oman
Mohanaad Shakir, University of Buraimi, Al Buraimi, Oman

ABSTRACT

A teaching approach plays an important role in teaching and learning process of an introductory programming (IP) course. The teaching approach should focus on different programming skills required by novice programmers. This study introduced the teaching and learning approach based on an ADRI (approach, deployment, result, improvement) approach in the IP course which focuses on both programming knowledge (syntax and semantics) and problem-solving strategies. The authors compared the teaching and learning approach of the IP course with the five levels of SOLO taxonomy. This paper assessed the students’ performance by using different assessment tasks based on the four stages of the ADRI approach. Results show that the current teaching and learning approach of the IP course addressed all the five levels of SOLO taxonomy. The students’ performance in the approach and result stages (82%) are good, and the performance in the improvement (71%) and deployment (69%) stages are satisfactory. Overall, the ADRI approach provides positive impact on the teaching and learning process of the IP course.

KEYWORDS

Learning to Program, Problem-Solving Skills, Programming Education, Pseudo-Code Technique, SOLO Taxonomy

INTRODUCTION

Learning to program is considered as a difficult area of learning for significant number of novice programmers (Malik et al., 2019a; Reardon & Tangney 2014; Shuhidan, 2012). One of the challenges for novices is to acquire a number of different skills at the same time. Novices have to learn not only the syntax and semantics of the programming language but also have to focus on the problem solving strategies. On the other side, Iqbal and harsh (2013) mentioned that introductory programming courses did not pay equal attention to programming knowledge (syntax and semantics) and problem solving strategies (pseudo-code and flowchart). De Raadt (2008) concluded that 6 out of 40 introductory programming textbooks incorporated problem solving strategies throughout the book. Robins et al (2003) discussed that ‘typical introductory programming textbooks devote most of their content to
presenting knowledge about a particular language’ (p. 141). Winslow (1996) discussed ‘that novice programmer knows the syntax and semantics of individual statements, but they do not know how to combine these features into valid programs’ (p. 17).

Papp-Varga et al. (2008) discussed that the teaching methodology of ICT teaching is not well-established as compared to other subjects such as mathematics or physics because it is comparatively a new problem domain. Consequently, different instructors are using their own blend of methods. Gomes and Mendes (2014) interviewed IP instructors and concluded that some instructors’ emphasized that a good choice of programming exercises and teaching materials also play an important role in teaching and learning process of the course.

This study compared a current teaching and learning approach based on the ADRI approach in the introductory programming (IP) course with SOLO taxonomy. The comparison with SOLO taxonomy helps us to determine whether the current teaching and learning approach provides all the necessary programming skills to the novice programmers. The students’ performance in the IP course was assessed based on the four stages of the ADRI approach.

This paper is organized into a number of sections starting with an introduction to the novice programmers and ADRI approach, and SOLO taxonomy, followed by a review of relevant literature. The research questions and methodology used are then described, and results are reported and elaborated. The paper concludes with a summary of the research outcomes.

NOVICE PROGRAMMERS

The Oxford English Dictionary (2014) defines novice as ‘a person new to and inexperienced in a job or situation’. Researchers defined novice programmers as those who are new, beginner or in the first stages of becoming a programmer (Thomas et al., 2004). In our context, novice is normally a first semester student within the Information Technology (IT) department and who has little or no prior experience or knowledge of programming. Generally, programming is considered an important component of computer science study plans and it can lead to a rewarding career (Robins at el, 2003).

It is generally accepted that novice programmers need 10 years of experience to become an expert programmer (Winslow, 1996). Dreyfus and Dreyfus (1986) proposed five stages for novices to become experts: novice, advanced beginner, competence, proficiency and expert. Winslow (1996) suggested that four year undergraduate degree graduates would rank between competent and proficient.

Some of the characteristics of novices drawn from various studies are:

- They have fragile knowledge which means they know the concepts but they fail to recognize when to use it (Perkins and Martin, 1986).
- They lack a mental model of the area (Kessler and Anderson, 1989).
- They emphasize surface knowledge of the subject (Winslow, 1996).
- They use general problem solving strategies rather than problem domain dependent strategies (Winslow, 1996).

Characteristics of Novice Programmers

The IP course is one of the first courses novices take in computer science education (Matthiasdottir 2006). Novices have to take it to progress in their study plan. Many face difficulties in the first few weeks of their programming course. They are unsuccessful when attempting to write computer code that meets stated objectives (Carbone et al., 2009). The whole situation de-motivates them and causes frustration which leads to disengagement with the curriculum. As a consequence, high failure and dropout rates are reported (Watson & Li, 2014; Zingaro, 2015).
Novices start programming with limited surface knowledge and skills. As a consequence, they generally approach programs ‘line by line’ rather than understanding the overall structure or ‘chunks’ of programs (Winslow, 1996).

They normally spend little time planning or testing code (Lane et al., 2012) and try to solve problems in the context of coding rather than understanding the broader sense of programs (Kölling and Rosenberg, 1996). Ala-Mutka (2003) mentioned that program planning is one of the important activities in learning programming languages. Further, novices spend more time learning programming language syntax as this is comparatively easy compared to developing the ability to write an algorithm that solves a given problem efficiently (Ansari, 2011).

Novice programmers’ general intelligence or mathematical abilities are also related to their success in learning programming languages (Bruce-Lockhart et al., 2000). Researchers also investigated that some programming language constructs are more difficult for novice programmers to understand than others. For example variable initialization is more difficult to understand than updating or testing variables (Soloway, et al., 1989).

Spohrer et al. (1986) suggested that some novices believe that programming is difficult because they need to grasp a significant amount of knowledge and different skills at the very beginning. Additionally, novices do not pay full attention to understanding tasks given by the instructor. This may lead them to a solution which does not meet the program specification (Ginat et al, 2004).

**ADRI APPROACH**

Malik and Coldwell-Neilson (2017a) proposed the ADRI approach in the teaching and learning process of the introductory programming (IP) course. The ADRI approach consists of four stages (Approach, Deployment, Result and Improvement). It is also considered as a well known quality assurance model in education and business sectors (Razvi et al. 2012). Australian, New Zealand and Omani universities are using the ADRI model in their quality audit process (Malik, 2016a) (Table 1).

The first stage (Approach) focuses on problem solving strategies such as pseudo-code and flowchart. The second stage (Deployment) deals with programming knowledge (syntax and semantics) of the programming language. The third stage (Result) focuses on input, output and process used to solve a given problem statement. Lastly, the fourth stage (Improvement) deals with different programming constructs by introducing new requirements in a given problem statement.

Table 2 depicts a programming example based on the four stages of the ADRI approach.

Table 1. Malik and Coldwell-Neilson (2017b) described the four stages of the ARDI approach

<table>
<thead>
<tr>
<th>Four stages of the ADRI model</th>
<th>Learning to Program task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>This stage deals with problem solving strategies such as pseudo-code and flowchart</td>
</tr>
<tr>
<td>Deployment</td>
<td>This stage focuses on syntax and semantics of the programming language</td>
</tr>
<tr>
<td>Result</td>
<td>This stage focuses on output, input and process used to solve a given problem statement.</td>
</tr>
<tr>
<td>Improvement</td>
<td>This stage deals with different programming constructs by introducing new requirements in a given problem statement.</td>
</tr>
</tbody>
</table>

The first section of the programming example in Table 2 shows a given problem statement. The second section (Approach) of the programming example covers problem solving strategies. Novice programmers have to solve the given problem statement by using pseudo-code and flowchart techniques. This section helps novices to focus on problem domain without much worry about the syntax of the programming language. Moreover, it also helps novices to understand the problem statement in a different way (draw flowchart diagram and write pseudo-code).
Table 2. Programming example based on four stages of the ADRI approach

<table>
<thead>
<tr>
<th>Step 1: Approach – Problem solving strategies</th>
<th>Pseudo-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Approach – Problem solving strategies</td>
<td></td>
</tr>
<tr>
<td>1. Start</td>
<td></td>
</tr>
<tr>
<td>2. Read Second, Minutes, Hour</td>
<td></td>
</tr>
<tr>
<td>3. Total_time_second = Second + Minutes * 60 + Hour * 3600</td>
<td></td>
</tr>
<tr>
<td>4. Print Total_time_second</td>
<td></td>
</tr>
<tr>
<td>5. Stop</td>
<td></td>
</tr>
</tbody>
</table>

Flowchart

Start

Read Second, Minutes, Hour

Total_time_second = Second + Minutes * 60 + Hour * 3600

Print Total_time_second

Stop

Step 2: Programming Knowledge

```cpp
#include <iostream>
using namespace std;

int main()
{
    int second, hour, minute, total_time_second;
    cout << "Enter the time in second.\"; 
    cin >> second;
    cout << endl;
    cout << "Enter the time in minutes.\"; 
    cin >> minute;
    cout << endl;
    cout << "Enter the time in hours.\"; 
    cin >> hour;
    cout << endl;
    total_time_second = second + minute*60 + hour*3600;
    cout << "The total time in second = " << total_time_second << endl;
    return 0;
}
```

Step 3: Result

- **Expected output:** enter the time in second: 65
- enter the time in minutes: 20
- enter the time in hours: 2
- The total time in second: 8465

**Process:** total_time_second = 65 + 20*60 + 2*3600

**Achieved output:**

- **Goals:** Achieved: ☐
  Not Achieved: ☐

**Syntax Error:** Incompatible data types / Initialization

Example:

```cpp
7 int second = "Hello";
```

Reason: integer value is expected in variable second on line 7. The string value (Hello) is assigned to integer variable.

Step 4: Improvement

Update above program that prompts the user to input seconds and calculate hour, minutes and remaining seconds.

**Expected output:** Enter the time in Seconds: 7502
- Hours = 2
- Minutes = 5
- Seconds = 2
The third section (Deployment) deals with programming knowledge (syntax and semantics) of the programming language. In this section, novices have to translate the given problem statement into a computer program by using C++ language. Moreover, novices can focus more on programming knowledge in this section because the logic to solve the given problem statement is already developed in the second section.

The fourth section (Result) covers input, output, a process to solve the given problem statement, and examples of common syntax errors in the programming domain. This section helps novices to understand the given problem statement in a better way. Moreover, it helps novices to understand the process used to solve the given problem statement. Examples of common syntax and semantics errors are presented in this section which helps students to avoid common compilation errors in the programming domain.

The fifth section (Improvement) covers new programming requirements which provide more practice and hands-on experience of different programming constructs to novices. This section extends the initial problem statement of first section with one or more additional requirements. This additional requirement helps novices to re-think about their solution which is already developed in the third section to incorporate new requirements. This process provides novices more practice and promotes deep learning of the programming domain. Moreover, it also helps novices to understand different programming constructs.

An ADRI editor based on the four stages of the ADRI approach was prepared to support the teaching and learning of the IP course as shown in Figure 1. There is a separate interface for each stage of the ADRI approach in the editor. Programming questions are embedded in the editor. Moreover, questions related to a particular topic can be accessed directly from the editor.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prestructural</td>
<td>No understanding – missed the point</td>
</tr>
<tr>
<td>Unistructural</td>
<td>Understanding of a component of relevant knowledge</td>
</tr>
<tr>
<td>Multistructural</td>
<td>Understanding of a few components of relevant knowledge</td>
</tr>
<tr>
<td>Relational</td>
<td>Able to relate multiple components of relevant knowledge</td>
</tr>
<tr>
<td>Extended abstract</td>
<td>Able to generalize set of knowledge to a new domain</td>
</tr>
</tbody>
</table>
SOLO TAXONOMY

In 1982, Biggs and Collis (1982) introduced the SOLO (Structure of the Observed Learning Outcome) learning taxonomy. The SOLO taxonomy provides a qualitative way to classify cognitive processes (Biggs and Collis, 1982). It mainly focuses on the content of learner’s responses in assessment tasks (Fuller et al, 2007). There are five categories in the SOLO taxonomy: prestructural, unistructural, multistructural, relational and extended abstract. Table 3 presents the categories used in the SOLO taxonomy and their descriptions.

The prestructural category is the lowest level where the learner shows no understanding of the topic. At the uni-structural level, the learner demonstrates little understanding of the topic. Next is multi-structural where the learner indicates understanding of a few components of the topic. The next two levels show the learner’s strong understanding of the topic. At the relational level, the learner can apply the acquired knowledge to a familiar problem or to a data set. The highest level, extended abstract, the learner is able to demonstrate application of knowledge to solve new problems or in new domains (Biggs and Collis, 1982; Biggs, 1999; Fuller et al, 2007; Shuhidan, 2012).

LITERATURE REVIEW

Various studies have been conducted to determine the impact of SOLO Taxonomy in programming courses. This section explores the literature and discuss different factors emerged after introducing SOLO Taxonomy in programming courses.

Castro and Fisler (2017) introduced a multi-faced SOLO taxonomy to track program design skills in an entire CS1 course. They described the calibration of SOLO taxonomy levels with programming skills as shown in Table 4. The first three levels (Prestructural, Unistructural and Multistructural) of the SOLO taxonomy cover syntactic understanding and the fourth level (Relational) requires semantic understanding of the programming domain. They omitted extended abstract as none of their students reached that level in this study.

Fuller et al (2007) concluded that the SOLO taxonomy encourages a holistic approach that supports deep learning. On the other hand, ‘there is not yet much reported experience of using it for assessment in a range of subjects’ (p. 156).

Lister et al (2006) analyzed novice programmers reading problems by using SOLO taxonomy. They suggested that students who couldn’t describe a short piece of code at SOLO’s relational categories were not intellectually well equipped to write similar code. Furthermore, they advocated a mix of reading and writing tasks in teaching and assessing novice programmers.

Clear et al (2008) analyzed 14 introductory programming students’ responses for three ‘explain in plain English’ questions by using the following four categories of SOLO taxonomy (Relational, Multistructural, Unistructural, and Prestructural). They found that it was not always straightforward to classifying them. They recommended three more categories in the SOLO taxonomy for
analyzing programming assessments. They called it ‘the final SOLO categories’ which included the following seven categories Relational, Relational Error, Multistructural, Multistructural Omission, Multistructural Error, Unistructural and Prestructural categories.

As discussed above, Shuhidan et al (2009) analyzed final exam questions of novice programmers by using SOLO and Bloom’s taxonomies. They classified questions with SOLO taxonomy where novices had to write ‘code to calculate the highest and lowest integer from a set of integers passed via the command line’ (p. 95). They found that to attain the relational classification level of SOLO taxonomy, novices should understand and connect different components of the solution for a given problem.

Corney et al (2011) conducted a study with novice programmers by using the SOLO taxonomy. They discovered in the third week of study that almost half of their sample students could not answer a simple question of swapping the values of two variables. They urged that novices started facing these problems from the beginning of the semester. They suggested incorporating new pedagogical approaches into the IP curriculum because the traditional programming exercises did not detect these problems until the end of semester.

RESEARCH QUESTIONS

It is evident from the previous section that SOLO taxonomy and ADRI approach can play an important role in improving the learning and teaching process. In this study, we propose two research questions to explore the benefits of using SOLO taxonomy and ADRI approach in the IP course.

The two research questions being addressed in this research study are as follows:

Q1: How the ADRI approach incorporates SOLO taxonomy in the teaching and learning process of the introductory programming course?

Q2: What is the impact of the ADRI approach in the introductory programming course?

Methodology

The first research question was probed by comparing the four stages of the ADRI approach with ‘Calibration of SOLO taxonomy with programming skills’ proposed by Castro and Fisler (2017) and shown in Table 4. The second research question was explored by analyzing the students’ performance in the IP course based on the four stages of the ADRI approach.

The students’ grades were collected anonymously from the registration department after obtaining an ethical approval from the Buraimi University College.

RESULTS

This section describes the results of this study and probes Research Questions 1 and 2.

SOLO Taxonomy and ADRI Approach

This section addresses the first research question which is:

Q1: How the ADRI approach incorporates SOLO taxonomy in the teaching and learning process of the introductory programming course?

The four stages of the ADRI approach were incorporated in the IP course by preparing the teaching materials based on them as shown in Table 2. All the lab exercises and examples related to
the teaching topics are prepared according to the ADRI approach. Moreover, the ADRI editor was prepared and introduced in the IP course to support the practical sessions.

Castro and Fisler (2017) calibrated the SOLO taxonomy with programming skills as shown in Table 4. The ADRI approach was introduced in the teaching and learning process of the IP course and discussed in the ADRI approach section. We extended ‘Calibration of SOLO taxonomy levels with programming skills’ table proposed by Castro and Fisler (2017) and added information which shows that how the ADRI approach addresses five levels of SOLO taxonomy. This extended information is shown in Table 5.

The four stages of the ADRI approach address all the five levels of SOLO taxonomy. The Approach stage of the ADRI covers problem solving strategies (pseudo-code and flowchart) which address the programming skills proposed by Castro and Fisler (2017) at Relational level of SOLO taxonomy. The deployment stage of the ADRI focuses on syntax which addresses the programming skills associated with Prestructural, Unistructural and Multistructural levels. The Result stage of the ADRI deals with input, output, and process used to solve a problem statement which addresses the programming skills related to Relational level. The last stage of the ADRI approach, Improvement covers new programming requirements and constructs and it is related to the programming skills associated with Extended Abstract.

### ADRI Approach in Introductory Programming Course

This section addresses the first research question which is:

**Q2:** What is the impact of the ADRI approach in the introductory programming course?

The ADRI approach was introduced in the teaching and learning process of the IP course. All the programming examples presented in the course were prepared based on the four stages of the ADRI approach. The ADRI editor was prepared to support the ADRI approach in the course. The ADRI approach was not only incorporated in the teaching and learning process of the introductory programming (IP) course but it was also included in the assessment tasks of the course. Different assessment tasks were prepared in the course to cover all the four stages of the ADRI approach as shown in Table 6.

For the Approach stage of the ADRI, questions such as converting the given problem statement into a pseudo-code or flowchart program, and writing an equivalent pseudo-code for the given
Table 6. SOLO Taxonomy, ADRI approach and Assessment tasks

<table>
<thead>
<tr>
<th>SOLO Taxonomy</th>
<th>Calibration of SOLO Taxonomy with programming skills (Castro and Fisler, 2017)</th>
<th>Four stages of the ADRI approach</th>
<th>Tasks in each stage of the ADRI approach</th>
<th>Different assessment tasks for the ADRI approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prestructural</td>
<td>Syntactic</td>
<td>Deployment and Improvement stages</td>
<td>Syntax, Semantics, Constructs</td>
<td>Write a program for the given problem statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Correct the errors in the given program.</td>
<td>Solve the given program with different programming constructs.</td>
</tr>
<tr>
<td>Unistructural</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multistructural</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td>Semantic understanding</td>
<td>Approach and Result stages</td>
<td>Flowchart, Pseudo code, Input, Output, Process used to solve a problem statement</td>
<td>Convert the problem statement into a flowchart. Write a pseudo code for a given problem statement. Write an equivalent pseudo code for the given flow chart program. Evaluate the output of the given programs. Calculate the output of the given code with specific input values. Identify and correct the errors in the given codes. Evaluate the given expressions.</td>
</tr>
<tr>
<td>Extended Abstract</td>
<td>Omitted because none of their students reached that level in this study</td>
<td>Improvement stage</td>
<td>New programming requirements and constructs</td>
<td>Solve the given program with different programming constructs. Solve the program given in one programming construct with other programming construct.</td>
</tr>
</tbody>
</table>
flowchart program are introduced in the assessment part of the IP course. The deployment stage includes questions such as writing a program for the given problem statement and correcting the errors in the given program. The Result stage covers questions such as evaluating the output of the given code, calculating the output of the given code with specific input values, identifying and correcting the errors in the given codes, and evaluating the given expressions. Lastly, the improvement stage includes questions such as solving the given program with different programming constructs, and writing the program given in one programming construct with other programming construct.

The students’ performance was calculated by analyzing their marks in the related questions to the four stages of the ADRI approach in the IP course as shown in Figure 2. Results show that students’ performance in the Approach and Result stages (82%) are good, and the performance in the Improvement (71%) and Deployment (69%) stages are satisfactory.

DISCUSSION

Table 5 shows that the four stages of the ADRI approach cover the five levels of SOLO taxonomy. The four stages of the ADRI approach focuses on different programming skills such as program design, program comprehension, and language features. The Approach and Result stages of the ADRI focus on semantic understanding. The Deployment and Improvement stages deal with syntax of the programming language. Therefore, the ADRI approach addressed the programming skills aligned with SOLO taxonomy by Castro and Fisler (2017). Moreover, it shows that SOLO taxonomy is not only meant for assessment, it can be also used in designing curriculum. This finding is consistent with Biggs and Collis (1982).

The ADRI approach promotes problem solving strategies and algorithmic thinking in the IP course. Students practice pseudo-code technique for all the questions presented in the course. This process helps students to understand the given problem statement in a better way without much worry about the syntax of the programming language. This approach is consistent with Mathew et al. (2019).
where they developed and offered PROSOLVE game in the IP course to promote problem solving skills. Moreover, algorithmic thinking is promoted in the IP course by introducing the ADRI approach (Malik et al. 2019b; Malik et al. 2019c). Some studies suggested that e-learning impacted positively on the process of learning (Tawafak et. Al. (2020); Alfarsi et al. (2020); Tawafak et. Al. (2019)).

The ADRI-based teaching materials are prepared for all the teaching topics of the IP course which promotes practice of problem solving strategies, syntax and semantic for novice programmers throughout the semester. Moreover, this process supports students to acquire deep-learning of the programming domain. This finding is consistent with Winslow (1996) who suggests that ‘if novices want to become an expert programmer they have to practice, practice and practice programming problems’ (p.142).

Ala-Mutka (2004) argued that learning to program consists of several activities such as ‘learning the language features, program design and program comprehension’ (p. 3). On the other side, the traditional approach used in the IP course emphasizes more on the syntax of the programming language. The four stages of the ADRI approach address this issue and focus equally on programming knowledge (syntax and semantic) and problem solving strategies. Novice programmers have to practice both programming knowledge and problem solving strategies in all the questions based on the ADRI approach. This process promotes language features, program design and program comprehension skills.

The ADRI editor also helps students in practicing different programming skills in the IP course. The editor presents a given problem statement as four stages of the ADRI approach which promotes deep learning and different aspects of programming skills to students. Furthermore, this process also discourages the programming shortcut (Problem Statement → Codes) and promotes the proper programming process (Problem Statement → Solution plans → Codes) suggested by Webster (1994). On the other side, each programming question in the editor consists of four steps which demand more time from novices to finish the exercises. This practice is challenging and boring for some novice programmers.

Different assessment tasks were introduced in the IP course related to the four stages of the ADRI approach which are calibrated with SOLO taxonomy as shown in Table 6. These tasks assess syntactic and semantics understanding of different programming concepts of students which promotes deep learning of the programming domain. Moreover, these tasks are intended to assess students learning outcomes of the course. This finding is consistent with Biggs (1999) which explains that SOLO taxonomy can also be used in designing the curriculum and setting level of learning outcomes of the course.

The analysis of students’ marks depict that the ADRI approach helps students in grasping the basic concepts of programming domain. The Approach stage of the ADRI presents problem solving strategy (pseudo-code) which helps students in understanding the given problem statement without much worry about the syntax of the programming language. This finding is consistent with Malik and Coldwell-Neilson (2018). Students’ performance in the deployment and improvement stages of the ADRI approach is satisfactory compared to the Approach and Result stages where the performance is good, one of the reasons for different results in stages of the ADRI approach is because the deployment and improvement stages deals with syntax of the programming languages. Writing a computer program for the given problem statement is one of the difficult tasks for novices in the programming domain. This finding is consistent with Winslow (1996) who discussed ‘that novice programmer knows the syntax and semantics of individual statements, but they do not know how to combine these features into valid programs’ (p. 17). On the other hand, the ADRI approach provides more practice of the syntax of the programming language to novices in deployment and improvement stages compared to the traditional teaching method for an introductory programming course (Malik et al., 2017c).
CONCLUSION

Different programming skills such as problem solving, critical thinking, syntax and semantics are required by novice programmers for learning to program. The current teaching and learning approach in the introductory programming (IP) course based on the four stages of the ADRI approach pay equal attention to problem solving strategies (pseudo-code and flowchart) and programming knowledge (syntax and semantics). The current teaching and learning approach based on the ADRI approach was compared with SOLO taxonomy in the IP course. The comparison helps us to determine whether the current teaching and learning approach provides all the necessary programming skills to the novice programmers. The students’ performance in the IP course was assessed based on the four stages of the ADRI approach.

The current teaching and learning approach based on the ADRI approach addresses all the five levels of SOLO taxonomy. The four stages of the ADRI approach focus on different programming skills such as program design, program comprehension, and language features. Moreover, the ADRI approach not only focuses on programming knowledge (syntax and semantics) but it also pays attention on problem solving strategies.

Different assessment tasks based on the four stages of the ADRI approach and aligned with SOLO taxonomy were prepared in the IP course. The students’ performance was calculated by analyzing their marks in the related questions to the four stages of the ADRI approach. Results show that students’ performance in the Approach and Result stages (82%) are good, and the performance in the Improvement (71%) and Deployment (69%) stages are satisfactory. The Improvement and Deployment stages require students to write a computer program for a given problem statement which is a challenging and difficult task for novices in programming domain. Overall, the ADRI approach helps students in grasping the basic concepts of programming domain.
REFERENCES


Sohail Iqbal Malik has been working as a head of department and an Assistant Professor at Information Technology Department, Buraimi University College, Oman since February 2007. He obtained his Ph.D. in Computer Science from Deakin University, Australia. His research interest includes: Algorithmic thinking, Computer Education, Technology in Education, and E-Learning.

Ragad M. Tawafak is a PhD candidate. Graduate from Mosul university in 1998. Complete her master from Mosul University in 2000. She is computer science lecturer. Working 4 years at Almustansriya University till 2004. Then working at Oman from 2004 till 2006 at Oman College for Technology and Management. She’s also working at Alburaimi University College from 2006 up to date. Her publications were published on Web of Science and Scopus journals. Her interesting areas are e-learning, face verification, social media, m-learning, and quality assurance.

Mohanaad Shakir is an Assistant Professor at University of Buraimi, Oman. He holds a B.S.C Degree in Computer Science from the University of Almamoon, Baghdad, Iraq; Post Diploma in Computer Security from University of Technology, Iraq and M.Sc. in Information Technology (MIT) from the University of Tenaga National(UNITEN), Putrajaya, Malaysia. He is a Ph. D. candidate in Information Communication Technology at the University of Tenaga National (UNITEN), Putrajaya, Malaysia. His research interests include Cipher Security, Computer-Aided Learning and Cloud Computing Security.