

# Constructing an Ontology-based Portfolio Assessment Scheme to Aid Novice Learners Learn Programming

Jui-Feng Weng<sup>a</sup>, \*Shian-Shyong Tseng<sup>b</sup>

<sup>ab</sup>*Department of Computer Science, National Chiao Tung University, ROC*  
sstseng@cis.nctu.edu.tw

**Abstract:** In recent years, the programming learning has been widely applied to support the training of the problem solving skill in universities and senior high schools. With our observations, there is a learning gap from a novice who can recognize the program statement to a well-trained programmer who can write a correct program to solve a given problem. The experience of problem solving is important for learners to understand the different expression of the program syntax. To aid the novice learner learn the programming, our idea is to use the mastery learning theory with “learning by example” strategy. The programs of classical problems are collected and novice learners are asked to trace the problem solving process of the programs. To collect the learning feedbacks from learners, the buggy patterns relating to different programming capabilities are embedded in the original sample programs to facilitate the error detection assessment. If the detection of the embedded bugs can not correctly done by novice learners, these symptoms are recorded as portfolio for further misconception analysis. Finally, the experimental result shows the findings of the proposed methodology.

**Keywords:** programming learning, misconception, ontology, learning by example

## Introduction

In recent years, the programming learning has been widely applied to support the training of the problem solving skill in universities and senior high schools. In traditional programming learning courses, novice learners take lots of time in the learning of program syntax. However, they still usually make mistakes while in programming due to having some misconceptions on the usage of program syntax for problem solving.

With our observations, the learning of program syntax is difficult because there is still a big gap from a novice who can recognize the program statement to a well-trained programmer who can write a correct program to solve a given problem. The Bloom’s taxonomy (Bloom, 1956) [2] can be used to classify the programming capabilities into six levels. The first is knowledge level in which the learner can recognize the syntax of program. The second is comprehension level in which the learner can trace the program and understand the semantic purpose of program. The third is application level in which the learner can understand the mapping from the principles to program code. The fourth is analysis level in which the learner can contrast the programs in cognitive level. The fifth is synthesis level in which the learner can rewrite the learned program to solve a new problem. The sixth is evaluation level in which the learner can evaluate the performance of the

program. In this paper, we aim to provide the programming learning assistance for the knowledge and comprehension levels.

The experience of problem solving is important for learners to understand the different expression of the program syntax. To aid the novice learner learn the programming, our idea is to use the mastery learning theory (Block, 1971) with “learning by example” strategy, where programs of classical problems are collected and novice learners are asked to trace the problem solving process of programs. Therefore, it can extend learners’ understanding of given programs for solving different problems. Since in the mastery learning, the learning goal for students should be well defined in advance, the taxonomy of programming learning capabilities, such as the detection of syntax errors, semantic errors, or logical errors, can be collected and constructed as the Program Capability Ontology (PCO).

However, for the sample programs provided for learners to learn, we still want to know if the learners can really trace the program or not. To reveal the learning feedbacks from learners, the buggy patterns relating to different programming capabilities are collected and embedded in the original sample programs to facilitate the error detection assessment. If the detection of the embedded bugs can not correctly done by novice learners, these symptoms are recorded as portfolio for further misconception analysis.

For the learners, the misconception may have some major symptoms and some minor symptoms. In order to assist the teacher in managing the relations of symptoms and misconceptions, the repertory grid with misconceptions as columns and symptoms as rows are used. However, it is not easy for teacher to fill in the symptoms of misconceptions completely. With the cognitive science, it is easier to figure out the attributes of objects by comparing three objects in each time. Thus, in the repertory grid approach, the teachers are asked to distinguish every three misconceptions by the symptoms iteratively. If any two misconceptions can’t be distinguished by the existing symptoms, new symptoms should be added. Therefore, the obtained repertory grid can be used to predict learner’s misconceptions from symptoms. Thus, the repertory-grid-based assessment approach can be applied to support the detection of the root cause of symptoms. Finally, the most possible misconception can be concluded and the remedial suggestion can be provided to support the learner’s learning.

## 1. The ontology-based portfolio assessment scheme

In this paper, our idea is to apply the mastery learning for novice learners with the “learning by example” strategy where learners can extend their understanding of program syntax by understanding different problem solving processes in the examples. However, it raises the following technical issues: how to define the knowledge structure of learning, how to collect the learners’ feedbacks, and how to provide assessment for learning guidance.

To solve the issues above, we proposed the ontology-based approach to organize the knowledge structure of programming capabilities and facilitate the error detection assessment, where the **Programming Capability Ontology (PAO)** as shown in Figure 1 is constructed according to the detection capabilities of the syntax error, misuse token, and logical error.

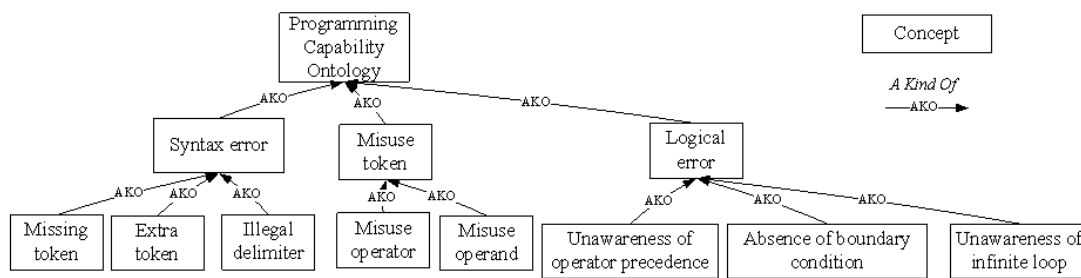


Figure 1: The Programming Capability Ontology

The explanations of PCO are given as follows.

**1). The detection of syntax error that violates the grammar of programming language.**

- **Missing token:** learner can detect the token omitted in the statement such as the missing token “;” omitted in the end of the statement “*printf("hello world!")*”.
- **Extra token:** learner can detect the redundant token which should be removed such as extra token “;” in statement “*for(...); printf(“\*”);*”.
- **Illegal delimiter:** learner can understand that delimiter is a sequence of one or more characters used to specify the start and end boundary between separate regions of program expressions such as the curly brackets “{“ and “}”.

**2). The detection of misuse token error that happened in the mapping of pseudo code to program.**

- **Misuse operator:** learner can detect the wrong usage of comparison operator “==” with assignment expression “=” in the statement “*if(a=1)*”.
- **Misuse operand data type:** learner can detect the incorrect operand usage with wrong data type or constraint such as integer variable “*a*” is mismatched with the wrong data type in the statement “*int a = ‘c’;*”.

**3). The detection of logical error that happened in the design of pseudo code for solving a given problem.**

- **Unawareness of operator precedence:** learner can detect the computational priority of operators such as the programming for formula “ $(a+b)*h/2$ ” may be written as “*Area\_Trapzium = a+b\*h/2;*” where the unawareness of operator precedence may cause the wrong result.
- **Absence of boundary condition of some variable:** learner can detect the boundary condition of variables such as the divided by zero error in statement “*b=0; a=1000/b;*”.
- **Unawareness of infinite loop:** learner can detect if the stopping criterion of “for/while” or “if” the branching condition of expression is unreachable such as the infinite loop in statement “*for(i=1, i<10, i--)*” or “*i=1; while(i>1){ i++; }*”.

## 2. Error detection assessment

Accordingly, the buggy patterns database can be constructed and indexed by the taxonomy of PCO. Thus, the required buggy patterns can be embedded in the selected programs and the novice learners are asked to detect those buggy patterns by tracing the program to facilitate the error detection assessment. Two examples of “*selection sort*” and “*factorial number generation*” programs with embedded buggy patterns are shown in Example 1.

### Example 1. Examples of program with embedded buggy patterns

With the constructed buggy patterns database, the buggy programs needed for error detection assessment can be provided. As shown in Figure 2, the example of *selection sort* with Program No. *Q001* is embedded with buggy patterns “*misuse token*” in line 5, the “*infinite loop*” in line 8, and the “*missing token*” in line 15; the example of *factorial number generation* with Program No. *Q002* is embedded with buggy patterns “*Misuse operand data type*” in line 7, the “*Misuse operator*” in line 10 and line 12.

<pre>// Program No.: Q001 // Problem Description: selection sort // Please write a program to sort the numbers in the input array. // Input: numerical array, array size // Output: the sorted array 1. void selectionSort(int numbers[], int array_size) 2. { 3.     int i, j; 4.     int min, temp; 5.     for (i = 0; i = array_size-1; i++) //bug: Misuse operator 6.     { 7.         min = i; 8.         for (j = i+1; j &lt; array_size; j--) //bug: Infinite loop 9.         { 10.            if (numbers[j] &lt; numbers[min]) 11.                min = j; 12.        } 13.        temp = numbers[i]; 14.        numbers[i] = numbers[min]; 15.        numbers[min] = temp //bug: Missing token 16.    } 17. }</pre>	<pre>// Program No.: Q002 // Problem Description // Please write a program to show the factorial number // Input: numerical number n // Output: the factorial number from 1 to n 1.#include &lt;stdio.h&gt; 2.#include &lt;stdlib.h&gt; 3.int main () 4.{ 5.    int i,n,sum=1 6.    printf("Input a integer, which will be used to calculate factorial\n"); 7.    scanf("%d",&amp;n); //bug: Misuse operand data type 8.    for(i=1;i&lt;=n;i++) 9.    { 10.        sum*i=sum; //bug: Misuse operator 11.    } 12.    printf("%d!=%d\n",n,sum); //bug: Misuse operator 13.    system("pause"); 14.    return 0; 15. }</pre>
---	---

Figure 2. Buggy programs of “*selection sort*” and “*factorial number generation*”

After the error detection assessment, the symptoms of failing to detect the buggy patterns are recorded in the portfolio. The data format of novice learner’s portfolio is defined as follows.

**Definition 1. The novice learner’s portfolio**

The portfolio of a novice learner is composed of a set of testing results:  $P = \{s_1, s_2, s_3, \dots\}$  where symptom  $s_i = (Program\_no, line\_no, position\_no, symptom\_id)$  represents the novice learner’s mistake on program detection. The *Program\_no*, *line\_no* and *position\_no* denote the position information of the occurred symptoms. The *symptom\_id* denotes the identification of buggy pattern.

**Example 1. The portfolio**

Assume that the learner failed to detect the “*misused operator*” or “*infinite loop*” buggy patterns in program Q001, then the portfolio is  $\{(Q001, 5, 10, Bug\_misused\_token), (Q001, 8, 1, Bug\_infinite\_loop)\}$ .

According to the above portfolio, the assessment of the misconception can be obtained by analyzing the symptoms in the portfolio. Since there may be many possible reasons for the symptoms, the repertory grid approach (Hwang & Tseng , 1990) is used to distinguish different misconceptions and identify the root cause by the proposed symptoms. As shown in Example 2, the first row of the Repertory grid represents the possible misconceptions and the first column represents the obtained related symptoms. The values of the table from 5 to 1 denote the relation of symptoms and misconceptions where 5 denotes “the most related” and the 1 denotes “not related”. The construction of repertory grid can be started from asking teachers to compare and distinguish three misconceptions in each time. It can proceed until relations of all symptoms and misconceptions are considered. Afterward, the

constructed repertory grid can support the misconception assessment from symptoms. The example of repertory grid is shown in Example 2.

**Example 2. The repertory grid approach for misconception assessment based on symptoms**

The Table 1 shows the syntax error symptoms:  $\{S_1, S_2, S_3\}$  and the possible syntax error misconceptions:  $\{M_1, M_2, M_3\}$ .

Table 1: Repertory grid for misconception assessment

<i>misconception</i> <i>Symptom</i>	M <sub>1</sub> :program structure	M <sub>2</sub> :unfamiliar the statement	M <sub>3</sub> :incautious
S1: Missing token	2	4	3
S2: Extra token	2	4	3
S3: illegal delimiter	5	4	2

Thus, the assessments of different symptoms are as follows.

- **Assessment based on symptoms of syntax error:** the symptoms of syntax error are including “missing token”, “extra token”, and “illegal delimiter”. The root causes for these symptoms can be most likely identified as misconception in “program structure” if the learner only has symptoms in illegal delimiter; “unfamiliar with the statement” if the learner fails in most of the symptoms; and “incautious” if the learner only has symptom in partial program.
- **Assessment based on symptoms of misuse token:** the symptoms of misuse token are including “misuse operator” and “misuse operand data type”. The root cause can be mostly identified as misconception in the “operator” if the learner only has symptom in program operator; “the operand data type” if the learner only has the corresponding symptoms; “fail in the mapping of pseudo code to program” if the learner has both symptoms.
- **Assessment based on symptoms of logical error:** the symptoms of logical error are including “unawareness of operator precedence”, “absence of boundary condition of variable”, and “unawareness of infinite loop”. The root cause can be mostly identified as the corresponding misconception only when learner passes the syntax assessment but has the logical error symptoms.

Therefore, if the major symptoms in novice’s portfolio matched the symptoms of repertory grid, it can be identified as major root cause misconception. Besides, cooperating with other minor symptoms, the possible misconceptions can also be provided to the learner for further remedial learning.

**3. The experiment and findings**

In the experiment, 20 freshman learners who participated in “Introduction to programming language” in the Asia University of Taiwan are involved. In the assessment, 11 programs are collected including 37 buggy patterns. After the analyzing of learners’ symptom portfolio, there are different misconceptions discovered as follows.

- Assessment for statement “printf”: There are 55% students with symptom on the detection of “printf”. With the consideration of more symptoms, 15% students almost failed to detect all bugs. Therefore, their misconception can be judged as *unfamiliar with the statement “printf”*. The other 40% students only failed in the misuse operator buggy pattern of “printf”. Therefore, their misconception can be judged as *misuse operator with the statement “printf”*.
- Assessment for statement “scanf”: There are 80% students with symptom on the

detection of “scanf”. With the consideration of more symptoms, 25% students almost failed to detect all bugs. Therefore, their misconception can be judged as *unfamiliar with the statement “scanf”*. The other 55% students only failed in the misuse operand data type of “scanf”. Therefore, their misconception can be judged as *misuse operand data type with the statement “scanf”*.

- Assessment for the statement “if”: There are 85% students with symptom on the misuse operator of “if”. With the consideration of more symptoms, 45% students failed to detect all bugs of the misuse operator. Therefore, their misconception can be judged as *misuse operator with the “if”*. The other 40% students also failed in the bugs of operator precedence, therefore, their misconception can be judged as *“unawareness of operator precedence”*.
- Assessment for the statement “for”: There are 65% students with symptom on the missing token of “for”. Since they only failed in this bug, therefore their misconception can be judged as *unfamiliar with the “for” statement*.

Thus, with the discovered misconception and symptoms, the feedback information can be provided to novice learners to facilitate the mastery learning with the original program examples.

#### 4. Conclusion

With the research we have done in this paper, the “learning by example” strategy is proposed to support the mastery learning of programming for problem solving. To make sure the learning status of the novice learners, the Ontology-based portfolio assessment scheme with the repertory-grid approach is applied to obtain the root cause of misconceptions from the symptoms of learners. Accordingly, the detailed remedial suggestion can be provided to support the further mastery learning with the original program example.

In the near future, more assessment approaches will be further applied with the “learning from imitation” such as program cloze testing, programming peer assessment for classic problems, etc. Furthermore, based on the knowledge structure of the program capability ontology, more misconceptions will be collected and classified to provide better assessment explanations.

#### Acknowledgements

This research was partially supported by National Science Council of Republic of China under the number of NSC95-2520-S009-007-MY3, NSC95-2520-S009-008-MY3, and NSC96-2522-S-009-002.

#### References

- [1] Block, J.H. (1971). Introduction to mastery learning: Theory and practice, in *Mastery Learning: Theory and Practice*, L. Block, P.W. Airasian, and J.H. Block, Editors. p. 2-28.
- [2] Bloom, B. S. (1956). *Taxonomy of educational objectives: The classification of educational goals: Handbook, I, cognitive domain*, New York, Toronto, Longmans, Green.
- [3] Xu, S. & Rajlich, V. (2004). Cognitive Process during Program Debugging, *Third IEEE International Conference on Cognitive Informatics*.
- [4] Hwang & Tseng (1990). EMCUD: A knowledge acquisition method which captures embedded meanings under uncertainty. *International Journal of Man-Machine Studies*. v33. 431-451.