A New Approach to Programming Language Education for Beginners with Top-Down Learning

http://dx.doi.org/10.3991/ijep.v3iS4.3216

Daisuke Saito, and Tsuneo Yamauara
Tokai University, Tokyo, Japan

***Abstract*—There are two basic approaches in learning new programming language: a bottom-up approach and a top-down approach. It has been said that if a learner has already acquired one language, the top-down approach is more efficient to learn another while, for a person who has absolutely no knowledge of any programming languages; the bottom-up approach is preferable. The major problem of the bottom-up approach is that it requires longer period to acquire the language. For quicker learning, this paper applies a top-down approach for a beginners who has not yet acquired any programming languages.**

***Index Terms*—Introduction: Bottom-up-approach, Learning-approach, Programming-language-learning, and Top-down-approach**

I. INTRODUCTION

A programming language education is one of the most serious issues in software development, and is quite time-consuming, in particular when targeting a person who has never learned any language before [1].

Usually, when learning a new programming language, a bottom-up approach is taken in many cases [2]. That is, first learning the data definition and grammar of the targeted language, and coding a sample program, then developing the actual software with a newly learned language.

Let's suppose the situation where an engineer with a full command of English must write a resume in French. In the bottom-up approach, he first learns the basics of French grammar, then takes the step of writing a resume in French. In the top-down approach, he gets "examples of French resumes," and changes some parts of the example to fit his purpose.

When beginners need linguistic acquirement, a bottom-up approach is used in many cases, and there are many case studies. As for a top-down approach, on the other hand, there is very little research in the programming learning method. And there are very few examples that can be used as a canonical programming.

II. LANGUAGE LEARNING APPROACH

There are two approaches in learning a programming language: a bottom-up approach (BUA) and a top-down approach (TDA).

A. *Bottom Up Approach(BUA)*

The bottom-up approach is a learning approach generally starting from the basics and moving to details. For a person who tries to learn C for example, he first studies grammar and data definition, then learns how to program. This approach is very frequently used in an educational institution and a book which teaches a programming language to beginners.

B. *Top Down Approach(TDA)*

The top-down approach mentioned in this paper is a learning method that first uses sample programming to acquire the language ability, then studying the details of the language, i.e., grammar and data definition. TDA approach uses sample coding. When a programmer learns C, for example, a simple and canonical sample program written in C is used, and he learns programming by understanding and changing the sample code.

TDA has many advantages over BU. Firstly TDA requires shorter period to acquire the programming language skill than BUA. TDA gives a final program, or a targeted goal to a learner and what he has to do is copy, think, change and verify. For the learner, each program presented to him gives a clear picture and goal of "This is the program that you will learn," and he can focus on a "block" of a program rather than a small piece of a program. He understands the program as a whole, a meaningful block of programming.

Since pieces of sample coding referred by the learner are picked up from the programs that furnish textbook-style and canonical programming style, he is expected that he will be able to simultaneously learn the coding style as well.

A typical learning process of TDA is, (1) read a block of source code with referring the comments, (2) guess and understand the meaning of a program sentence. The repetition of this process surely increases the program understandability, or the ability to understand someone else's programs.

Another advantage that a learner can enjoy from TDA is that he can acquire the sense of reusability, i.e., increasing the productivity by applying the copy-and-paste-based developing methodology. The software reusability is supposed to be one of the most promising remedies to improve the development productivity, which has only showed quite sluggish improvement for the last 4 decades. The basic idea of TDA is "using and changing the already existing programs," and is very similar to that of reusability.

The primary shortcoming of TDA, however, is that a learner can easily catch the overview of the program, but may not understand the details [2]. We named this mentality the ITIUE-syndrome (I Think I Understand Everything syndrome), and addresses this issue lately.

C. *Applicability of TDA*

TDA is expected to have many advantages, but is scarcely seen in the actual programming language education in particular when targeting a first-time learner. A rare case that TDA is used is in the situation where an advanced programmer, i.e., a software engineer who has a full command of one or more programming language, learns second or more language.

The major issue that prevents the wide dissemination of TDA even among advanced programmers is the difficulties in selecting sample programs to be used for TDA, i.e., the sample programs must be pragmatic, well-written, and obeying the programming style.

Based upon our teaching experience of programming languages, we tended to develop the confidence that TDA can be applicable to the novice learner's language learning, and conduct TDA is applicable in the programming language education targeting a first-time learner.

- TDA is applicable in the programming language education targeting a first-time learner.
- The first-time learner can enjoy the advantages of TDA.

III. OVERVIEW EXPERIMENT

The overview of the experiments to measure the effectiveness of TDA for the first-time learner is as follows:

A. *Select language*

C was selected as the target programming language.

B. *The detaile of laeners*

Six students (BU1, BU2, BU3 and TD1, TD2, TD3) were picked up, and BU1, BU2, BU3 were grouped for BUA while TD1, TD2, TD3 were for TDA. The details of six students are:

- BU1: An 18-year-old freshman majoring Information Engineering. No programming experience.
- BU2: A 23-year-old senior majoring electronics Eng. Almost no programming experience.
- BU3: A 23-year-old senior majoring Information Eng. Has experience of Jave programming only.
- TD1: An 18-year-old freshman majoring Information Engineering. No programming experience.
- TD2: A 23-year-old senior majoring Material Engineering. No programming experience.
- TD3: A 23-year-old junior majoring Psychology. No programming experience.

C. *Programming Sentences to Be Learned*

Six students were divided into 2 groups, i.e., the Three BUA learners and Three TDA learners, and were expected to learn the following 3 sentences:

- if Sentence
- for Sentence
- while Sentence

D. *Pre-Education*

Prior to the experiments (or, a programming contest), both groups received the pre-education as follows:

1) *BUA Group*

Three tutors, or programming experts, with using textbooks, taught 3 BU learners the basics of C including how to use variables and grammar of "if," "for" and "while." The tutoring period was approximately 90 minutes a person.

2) *TDA Group*

No tutors were assigned to TDA group (no advice or guidance were given), instead only the following 3 sample programs were showed:

```
#include <stdio.h>

int main(void){

        //Define "inputted number to be judged"
        int num;

        printf("Please enter an integer:");

        / A value is inputted
        scanf("%d",&num);  /

        // Judge if the value is even or odd //
        if((num%2)==0){
                printf("an even number");
                                }
        else{
                printf("an odd number");
        }
}
```

Figure 1. A Sample Program for "if-sentence"

```
#include <stdio.h>

int main(void){


        int i;              // Define "loop counter"

        // Define "total number," and set zero.
        int sum=0;

        // Repeat i times
        for(i=1;i<=10;i++)
        {
        sum=sum+i;     // Add " i" to the sum
    }
    printf("Total is% d",sum);
    //Display the value of the sum
}
```

Figure 2. A Sample Program for "for-sentence"

```
#include <stdio.h>
int main(void){

            int i=1;                // Define "loop counter"

            // Define "total number," and set zero.
            int sum=0;

            // Repeat while "i" is ten or less.
            while(i<=10){

                        // Add " i" to the sum
                        sum=sum+i;

                         // Add 1 to the loop counter.
                        i++;
    }

    printf("Total is% d",sum)
     // Display the value of the sum
}
```

Figure 3. A Sample Program for "while-sentence"

The program in Fig.1 judges if the inputted value is odd or even. The program in Fig.2 calculates the total number from 1 to 10 with using an "if statement," and Fig.3 does the same thing with a "while-statement." When 3 TDA learners received the above 3 programs, they tried to understand the meanings of each sentence by comparing the coding and comments, and repeated "change-and-run the programs" to verify that their understanding was correct.

E. Specifications of the Program to Be Developed

After 2 groups completed 90-minute learning, the following simple specification was presented to measure the effectiveness of TDA:

Make a "guess the secret number" program that satisfies the following requirements:

1. Get an inputted "guessed number."
2. A player can try 10 times.
3. Compare the guessed number and the secret number.
4. If the guessed number is not right, display a message like "The guessed number was larger (smaller). You can try 7 times."
5. If the guessed number is correct, end the program.
6. Set "256" as the secret number.

Figure 4. Specification of the Problem to Be Developed

In order for the 6 participants to make the required program, 30 minutes were given as their first trial of programming, but, as we anticipated, none of 6 participants have come out a correct one (a few of them were close, but were not good enough). We then gave a few clues (we gave more clues to the BUA group than the TDA team) and another 30 minutes.

IV. RESULTS, ANALYSIS AND INTERPRETATION

A. Results

TABLE1 shows the overall results of this programming experiment. "Minutes to code" is the duration time needed to code in minutes. "Correctness" is whether the program is functionally correct or not, or if the program works as defined in the specification (we applied 5-point grading system where, from 5 to 1, grading changes from excellent to poor). "Structure" refers to the program structure, and evaluates if the program follows the programming style including logic structure, data structure, indentation, naming conversion etc [3]. "Comments" evaluates whether or not comments are properly attached [4]. The "LOC" stands for "Lines of Code," or a number of program lines. The "Grammar" means whether the program is grammatically correct or not, or a participant can code the program without compilation errors within the given time.

We also conducted an interview with 6 participants to analyze the "program understandability" and "motivation,"

TABLE I.
PROGRAMMING RESULTS

BUA / TDA	**BUA**			**TDA**		
Participants	***BU1***	***BU2***	***BU3***	***TD1***	***TD2***	***TD3***
minutes to code	51	60	24	32	29	26
Correctness	4	4	5	4	4	4
Structure	5	5	5	5	5	4
Comments	no	No	yes	yes	Yes	yes
LOC	31	30	29	36	34	33
Grammar	5	5	5	5	5	5

B. Analysis

Here is one thing that we have to consider when the results are analyzed: BU3 was selected because he has the programming experience of Java (not C) only, but he extended his knowledge and experience to C programming, and showed quite high performance as is illustrated in TABLE1. We assumed that BU3 was an odd one to exclude from the results.

1) Minutes to Code

If BU3 is excluded because he is a semi-expert of Java, or an "odd man out," the learners of BUA takes nearly 100% more than the TDA learners to develop the program. Please note that, prior to the experiments, the 3 BUA learners took a C programming language lesson in a man-to-man fashion while the 3 TDA was given only sample programs, i.e., they acquired the programming ability by self-learning.

2) Correctness

All the programs made by 6 people worked as specified in the requirement specification illustrated in Fig.4. Thus, in terms of the functional quality, we do not see any differences between BUA and TDA, or another interpretation is "Shorter learning period did not bring poorer quality."

3) Structure

"Structure" here means whether or not the developed program follows the coding rules. Even if BU3 is excluded as an "odd man out" same as in "Minutes to

Code," the learners of BUA showed (slightly) better performance than the TDA people. This is the only instance that did not go as we anticipated before we conducted the experiments. We had an intensive interview with TD3 to figure out that he did not like programming (or, he tended to think that he was forced to joine the experiments), and reiterated copy-and-paste without any deeper consideration. It would be better, same as BU3, to assume TD3 as an "odd one out."

4) Comments

Same as in "Minutes to Code," if BU3 is excluded as an "odd man out," the learners of BUA did not write any comments while 100% of the TDA learners put comments. The TDA people wrote comments, because the TDA people always saw the well-commented program, and assumed that comments are must. The TDA-styled learning is also good for better program readability.

5) LOC (Lines of Code)

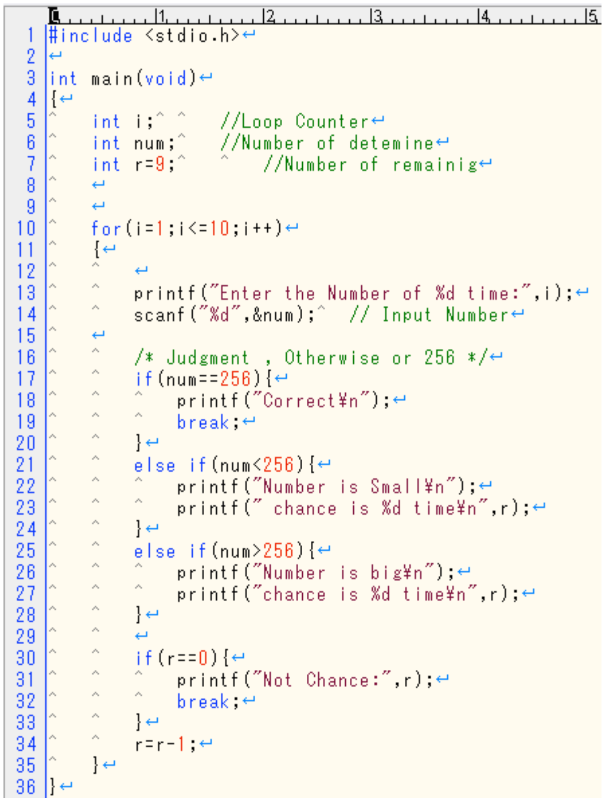
In terms of the number of program lines developed by 6 learners, there were not significant differences between the BUA learners and the TDA people.

6) Grammar

We checked grammatical errors that were rejected by C compiler: none of 6 programs had uncompilable errors.

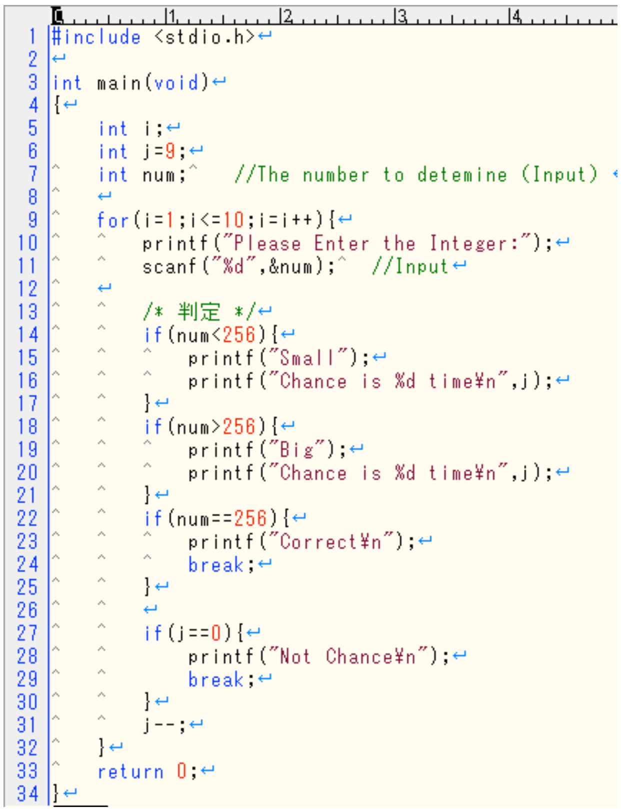
C. Further Analysis

Here we develop further analysis. Fig. 5 through Fig. 10 illustrate the source programs that were actually made by 6 examinees of the TDA group and the BUA group (Comments and messages in the source code were translated in English for readers' better understanding).。



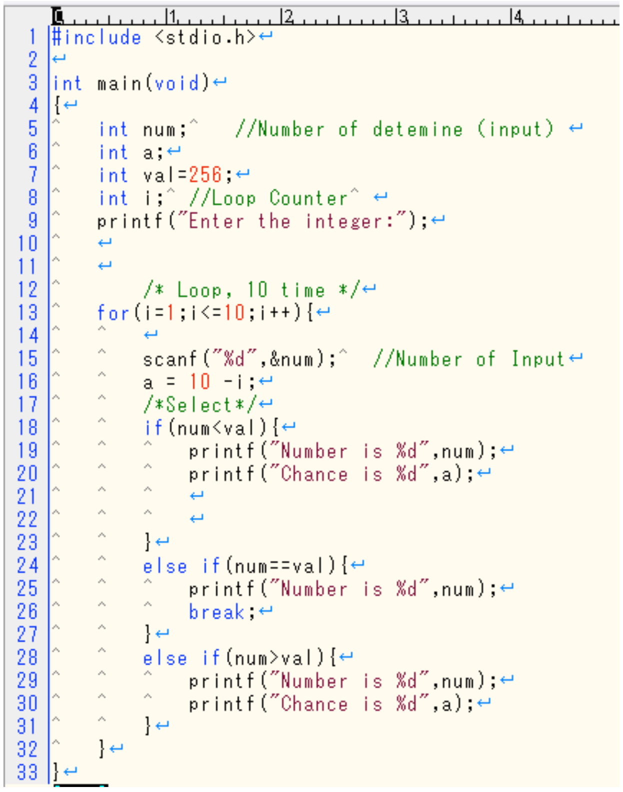
```
#include <stdio.h>

int main(void)
{
    int i;      //Loop Counter
    int num;    //Number of detemine
    int r=9;        //Number of remainig


    for(i=1;i<=10;i++)
    {

        printf("Enter the Number of %d time:",i);
        scanf("%d",&num);   // Input Number

        /* Judgment , Otherwise or 256 */
        if(num==256){
            printf("Correct¥n");
            break;
        }
        else if(num<256){
            printf("Number is Small¥n");
            printf(" chance is %d time¥n",r);
        }
        else if(num>256){
            printf("Number is big¥n");
            printf("chance is %d time¥n",r);
        }

        if(r==0){
            printf("Not Chance:",r);
            break;
        }
        r=r-1;
    }
}
```

Figure 5. TD1 Result Code



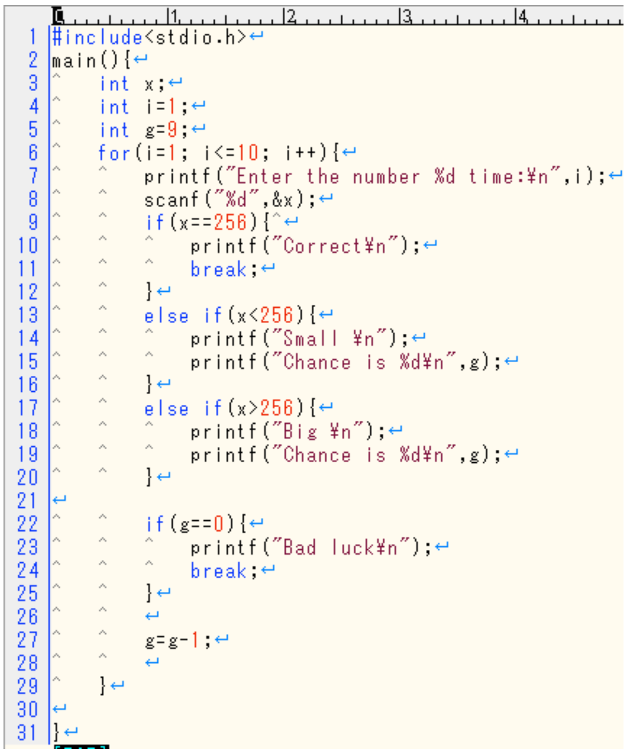
```
#include <stdio.h>

int main(void)
{
    int i;
    int j=9;
    int num;    //The number to detemine (Input)

    for(i=1;i<=10;i=i++){
        printf("Please Enter the Integer:");
        scanf("%d",&num);   //Input

        /* 判定 */
        if(num<256){
            printf("Small");
            printf("Chance is %d time¥n",j);
        }
        if(num>256){
            printf("Big");
            printf("Chance is %d time¥n",j);
        }
        if(num==256){
            printf("Correct¥n");
            break;
        }

        if(j==0){
            printf("Not Chance¥n");
            break;
        }
        j--;
    }
    return 0;
}
```

Figure 6. TD2 Result-Code



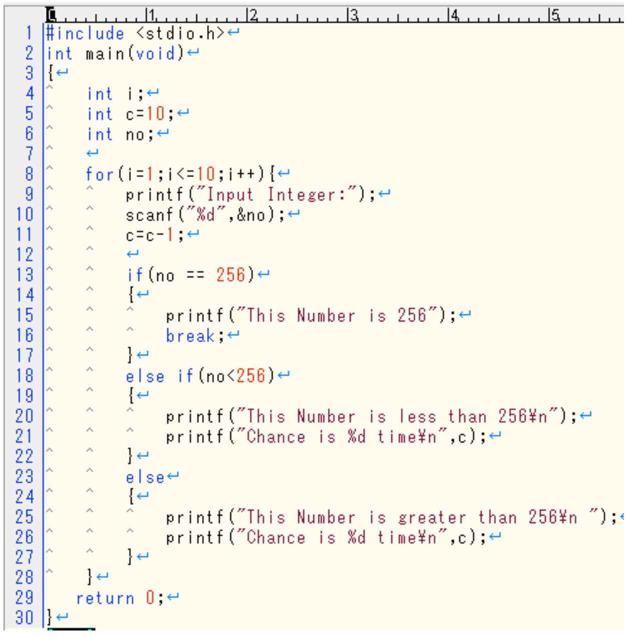
```
#include <stdio.h>

int main(void)
{
    int num;    //Number of detemine (input)
    int a;
    int val=256;
    int i;  //Loop Counter
    printf("Enter the integer:");


        /* Loop, 10 time */
    for(i=1;i<=10;i++){

        scanf("%d",&num);   //Number of Input
        a = 10 -i;
        /*Select*/
        if(num<val){
            printf("Number is %d",num);
            printf("Chance is %d",a);


        }
        else if(num==val){
            printf("Number is %d",num);
            break;
        }
        else if(num>val){
            printf("Number is %d",num);
            printf("Chance is %d",a);
        }
    }
}
```

Figure 7. TD3 Result Code



```
#include<stdio.h>
main(){
    int x;
    int i=1;
    int g=9;
    for(i=1; i<=10; i++){
        printf("Enter the number %d time:¥n",i);
        scanf("%d",&x);
        if(x==256){
            printf("Correct¥n");
            break;
        }
        else if(x<256){
            printf("Small ¥n");
            printf("Chance is %d¥n",g);
        }
        else if(x>256){
            printf("Big ¥n");
            printf("Chance is %d¥n",g);
        }

        if(g==0){
            printf("Bad luck¥n");
            break;
        }

        g=g-1;

    }

}
```

Figure 8. BU1 Result Code



```
#include <stdio.h>
int main(void)
{
    int i;
    int c=10;
    int no;

    for(i=1;i<=10;i++){
        printf("Input Integer:");
        scanf("%d",&no);
        c=c-1;

        if(no == 256)
        {
            printf("This Number is 256");
            break;
        }
        else if(no<256)
        {
            printf("This Number is less than 256¥n");
            printf("Chance is %d time¥n",c);
        }
        else
        {
            printf("This Number is greater than 256¥n ");
            printf("Chance is %d time¥n",c);
        }
    }
   return 0;
}
```

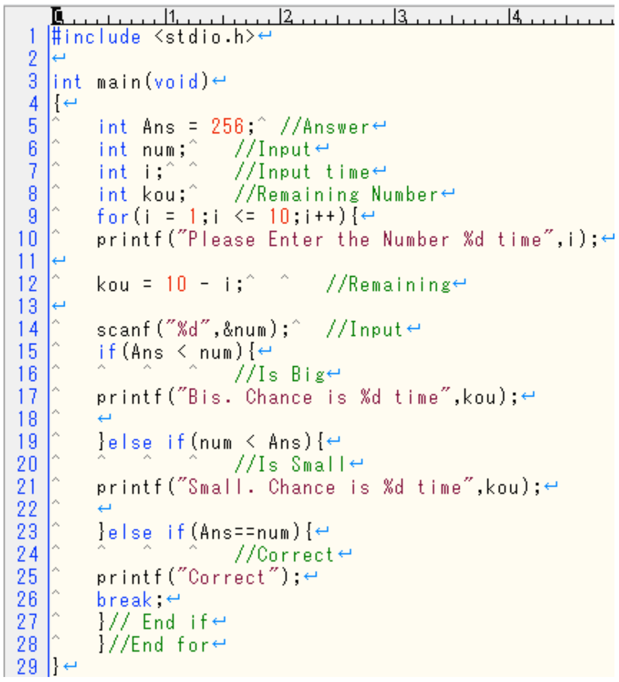
Figure 9. BU2 Result Code

1) *Quality*

We compared the source code of TDA and BUA to figure out little differences. The differences were very little but obvious in "comments" and "naming."

a) *Comment*

Only one person of the BUA group described comments while all the three of the TDA group added comments. This is because (1) all the sample programs had comments, the TDA examinees assumed that a source program must have comments, (2) we put emphasis on the importance of the comments prior to the programming, (3)



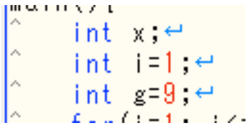
```
#include <stdio.h>

int main(void)
{
    int Ans = 256;  //Answer
    int num;    //Input
    int i;      //Input time
    int kou;    //Remaining Number
    for(i = 1;i <= 10;i++){
    printf("Please Enter the Number %d time",i);

    kou = 10 - i;       //Remaining

    scanf("%d",&num);   //Input
    if(Ans < num){
                //Is Big
    printf("Bis. Chance is %d time",kou);

    }else if(num < Ans){
                //Is Small
    printf("Small. Chance is %d time",kou);

    }else if(Ans==num){
                //Correct
    printf("Correct");
    break;
    }// End if
    }//End for
}
```

Figure 10. BU3 Result Code

we asked the TDA examinees to understand the meaning of the source code by comparing each program sentence and the attached comment. For the BUA learners, commenting or not commenting heavily depends on the educators: whether or not he teaches the importance of comments.

b) *Naming variables*

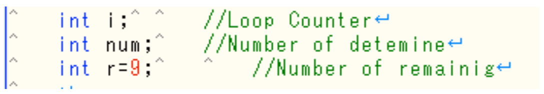
Between the TDA and BUA learners, there were significant differences in the "naming convention," or naming rules. Fig.11 illustrates examples of the variable names made by a BUA examinee. The variable names do not show or suggest the actual meanings of the variables.



```
    int x;
    int i=1;
    int g=9;
```

Figure 11.

Fig.12 shows an example of the variable names used by a TDA learner. Compared to the names of the BUA examinees, the variable names have some meanings, and suggest the usage of the variables. These better naming came from the sample source code that applies textbook-style naming convention.



```
    int i;      //Loop Counter
    int num;    //Number of detemine
    int r=9;        //Number of remainig
```

Figure 12.

There is a tendency that put more of a group of TDA is to understand the Naming of Things also when you look at the code of the other.

2) *Time*

In terms of the duration time to develop, even though there are individual differences, the TDA learners tended to spend shorter time than the BUA examinees as shown in Tbl 1.

D. *Interpretation*

1) *Program Understandability*

We conducted an interview with BUA people to find out that they (except for BU3) even did not have the sense of the programing understandability. This is because they wrote a program, but did not read it. On the other hand, the TDA learners started from reading programs with referring to comments. This unconsciously increased the learners' program understandability.

2) *Motivation*

Retaining motivation is extremely important and difficult when learning something, and many studies have addressed this issue of programming language learning [5] [6].

The interview revealed that TDA learners could retain the motivation much longer than the BUA people. The TDA learners clearly understood their goals because samples were given. People can retain motivation if a goal is showed.

V. ISSUE OF TDA

TDA has some issues: When we conducted the interview, the three participants of the TDA group mentioned the following comments:

- "I think I understood everything"(actually he did not)
- "I understood the process" (actually he does not know how to use it)
- "Because I see the goal, I thought it must be easy to overcome the problem"

We suspect that this is an undesirable side effect of TDA, and we name it ITIUE-syndrome (I Think I Understood Everything syndrome). Since the basic idea of TDA is, firstly, TDA clearly shows where to start and its goal by sample programs, then a learner studies programming, he tends to think that "I think I can easily make it," or in the worst case, "I completely understood everything" even though what he has to learn is how to actually and practically program.

This mentality must be corrected. One of the possible remedies will be to provide a program with some blank lines, or a "fill-in-the-blanks" type of programming questions to see how deeply the learner understands.

In addition, one must also consider the sample code to be provided. The sample code must furnish canonical coding style, naming convention, and algorithm so that a learner assumes the code as a textbook. We need criterion for choosing the sample source code.

VI. FUTURE WORK

Our experiments showed that TDA is quite effective for a beginner education. In order for us to analyze the more details of TDA, in particular the applicability and relationship between TDA and a programming language, we will conduct a larger scale programming experiments.

One of the most crucial issues that we have to tackle is that how we will be able to furnish canonical or textbook-styled programs which perfectly follow programming rules, use good algorithm, have proper comments, etc.

Also we will have to come out a language learning process based upon TDA to generalize the language education.

VII. CONCLUSION

When learning a new programming language, those who are familiar with programming languages tend to apply top-down approach or TDA, i.e., firstly attempting to understand program sentences then moving to grammar and details. Although this approach will surely bring a quicker learning compared to a bottom-up approach, or BUA, which starts from grammar then goes to program sentence, many education organizations for programming language have supposed that TDA was not good for beginners.

We picked up 6 students, who have never learned any programming language, and applied a top-down approach in their new programming language learning education.

The results revealed that TDA is quite effective even when a beginner is targeted: although there is no noticeable improvement in program writing between TDA and BUA, the learners in the TDA group significantly increased the program understandability (i.e., understanding other person's programming) because they reiterated "read and understand the sample programs. They also acquired the sense of reusability.

ACKNOWLEDGEMENT

We thank Hiroki Hayashi, Sho Takahashi, Yuta Saito, Ryo Masuoka, Hiroto Baba, Hirotaka Suma who participated into our programming experiments. Also we thank Daisuke Moriya, Yuya Hukuyama, Tomoyasu Tanaka for conducting the experiments, summarizing the data, and proof reading.

REFERENCES

[1] Lahtinen, E., Ala-Mutka, K., and Järvinen,, A study of the difficulties of novice programmers, ACM SIGCSE Bulletin, Sept 2005, Volume 37 Issue 3, pp. 14-18

[2] Margaret M. Reek, A TOP-DOWN APPROACH TO TEACHING PROGRAMMING, ACM SIGCS, Mar. 1995, pp. 6-9

[3] Brian W. Pike, Rob Kernighan, Practice of Programming, The (Addison-Wesley Professional Computing Series), Addison-Wesley Professional, Feb. 1999

[4] Xuefen Fang, Quality Software, 2001. Proceedings.Second Asia-Pacific Conference on, 2001, pp.73 – 78

[5] Tony Jenkins, The motivation of students of programming, ACM SIGCSE Bulletin, Sept. 2001, Volume 33 Issue 3, pp. 53-56 http://dx.doi.org/10.1145/507758.377472

[6] Takemura, Y.; Nagumo, H.; Nitta, N, Work in progress: Analysis of the relationship between teaching contents and motivation in programming education, Frontiers in Education Conference (FIE), Oct. 2012, pp. 1 – 2

AUTHORS

D Saito is with the Information and Communication Technology Department, Tokai University (e-mail: 3bjnm007@ mail.tokai-u.jp).

T Yamaura is with the Information and Communication Technology Department, Tokai University (e-mail: yamaura@keyaki.cc.u-tokai.ac.jp)

This article is an extended and modified version of a paper presented at the 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE2013), held 26-29 August 2013, Bali Dynasty Resort, Kuta, Indonesia. Submitted 30 September. Published as re-submitted by the authors 01 December 2013.