Positive Side Effects of Programming Classes — Especially on Thinking Ability and Mathematics —

Yukio IDOSAKA[1], Hiroto ASHIKAGA[2], Shuji KUREBAYASHI[3], Toshiyuki KAMADA[4], Susumu KANEMUNE[5], and Yasushi KUNO[6]

[1] Iinan Junior High School, Matsusaka City idosaka@gmail.com
[2] Tottori Technical Highschool ashi@logob.com
[3] Shizuoka University eskureb@ipc.shizuoka.ac.jp
[4] Aichi University of Education tkamada@auecc.aichi-edu.ac.jp
[5] Hitotsubashi University kanemune@acm.org
[6] University of Tsukuba kuno@gssm.otsuka.tsukuba.ac.jp

Abstract. From our teaching experiences of programming classes in junior high school, we believe that learning programming have positive effects on many area other than programming and computers itself. In this paper, our interim attempt to verify the above belief is presented, along with the overview of our programming classes using "Dolittle" educational object-oriented language. We have used three evaluation scheme, namely: (1) subjective analysis from the teacher's observation, (2) subjective enquiry to the students, and (3) evaluatin through paper tests. We report the former two in this presentation, whose results were quite positive. Those results suggests that students' thinking abilities, concentration and understanding of some mathematics concepts (XY-coordinates and figures) were enhanced by the programming classes.

1 Introduction

We have a hypothesis that programming education in school classes have various positive effects on students' abilities other than programming itself. The hypothesis is based on many years' teaching experiences on junior high-school by one of the authors (called "the teacher" hereafter).

The teacher has about 10 years' experiences of programming education in the subject "technologies and home economics" in Japanese junior high-schools. The aim was to let students gain experiences on what the principles of various softwares are and how they are crafted.

During the period, he started to feel that programming experiences not only support students' understandings on principles of software and computer, but also have positive effects on students' thinking abilities and creativity.

Therefore, we are now studying effects of programming education in the area other than programming ability itself. In school year 2005, we have conducted enquiries on how students feel about the various effects of programming classes.

In school year 2006, we have conducted comparative study with objective tests. In this paper, we report our interim results and plans for further research.

In section 2, overview of our programming classes is presented. In section 3, we report the teacher's subjective evaluation on positive effects of programming classes. In section 4, we report the evaluation results based on enquiries to the students. Finally in section 5, discussions and future plans are presented.

2 Overview of The Programming Classes

At Iinan junior high school (Matsusaka city, Mie prefecture, Japan), in the subject of technology and home economics, the teacher has been conducting 12 to 16 hours of programming classes for 3rd grade students every year (an exception was classes described in section 5, in which experiments were conducted against 2nd grade students).

We are using educational programming language Dolittle[1] in our classes. In Dolittle, Japanese characters are extensively used for operation names and identifiers (for Japanese version; English and Korean versions are also available). Therefore, Dolittle programs are easy to read and comprehend for students. Object-Orientation (O-O) is also extensively used in Dolittle[2][3]. Our curriculum contsists of three sections, namely: (1) turtle graphics, (2) timers and animations, and (3) GUI buttons and interactions, which we discuss below. Four hours (1hour = 50min) were assigned to each of those sections.

2.1 Turtle Graphics

In this unit, students learns basics of Dolittle language and Dolittle environment using turtle graphics.

At the first stage, "forward" and "rightturn" commands are used to draw a triangle, and then simple loop command "repeat" is introduced. Then, figure objects are created from turtle traces ("makefigure") and how they can be coloured ("paint") or moved ("moveto") freely are taught. Fig.1. shows one of sample programs, in which a triangle is drawn using a loop, the triangle is converted to figure object, coloured, and moved to the specific position.

```
ayumi = turtle ! create.
tri = [ayumi ! 100 forward 120 rightturn] ! 3 repeat makefigure (red) paint.
tri ! 100 100 moveby.
```

Fig. 1. A Dolittle program for drawing and positioning a figure

The goals of this unit is to teach the followings:

(1) A program is executed in line-by-line, step-by-step manner.
(2) A sequence of step can be repeated using a loop command.
(3) A programmer can create an object; an object accept commands.

Additionally, as a part of graphics, we also taught external angles (of triangles and squares), XY coordinates, distance of movement, and rotational degree.

At this stage, students could combine triangles, squares, circles or such to create their own pictures, as shown in Fig.2.

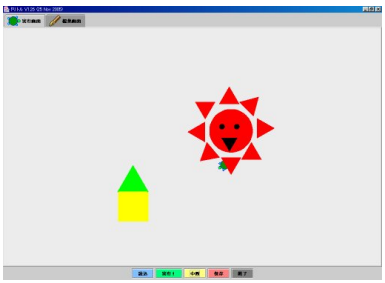

Fig. 2. A Screen for a student's drawing

2.2 Timer and Animation

In this unit, we taught timer objects. A timer object invokes a sequence of steps repeatedly, in a specified pace (one can specify delay between invocation and invocation count). With timer objects, one can move figures for some determined period to create animation, as in character movements in a video game. Fig.3 shows an example animation program (of rotating turtle and sliding figure).

```
ayumi = turtle ! create.
tri = [ayumi ! 100 forward 120 rightturn] ! 3 repeat makefigure (red) paint.
clock = timer ! create.
clock ! 1 duration 10 seconds.
clock ! [ayumi ! 15 rightturn. tri ! -10 -10 moveby] execute.
```

Fig. 3. An example animation program using a timer

At this stage, students observed movement of figures on the screen, and could grab images of rotational movement and sliding movement. After this unit, students could kraft animation programs with various movement of figures, as shown in Fig.4.

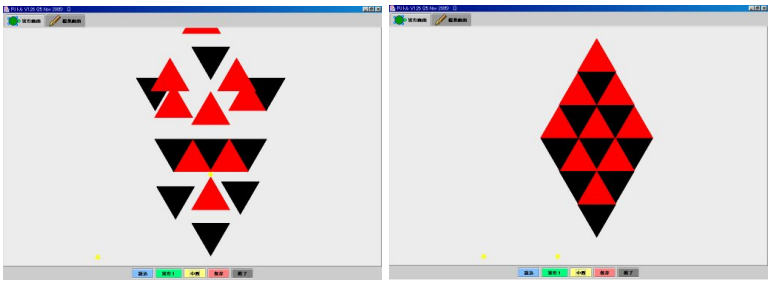

Fig. 4. Animation program crafted by a student

2.3 GUI Buttons and Interactions

In this unit, we taught the most primitive of GUI widgets — buttons. A button object can be created on the screen, can be positioned arbitrarily with specified label, and a sequence of steps can be specified as its action. During program execution, when the user press the button, that action is performed. Actually, we have taught students how to define "action" method for a button. Additionally, we taught XY-coordinates more thoroughly, to specify positions of the figures.

Fig.5 shows an example program in which the turtle moves (draws a line segment) each time the button is pressed.

```
ayumi = turtle ! create.
btn1 = button ! "forward" create.
btn1 ! -200 100 moveto.
btn1:click = [ayumi ! 50 forward].
```

Fig. 5. A Dolittle program which uses a button for turtle control

With buttons, students could develop their own "drawing software," in which a turtle can be moved according to button presses, figures created from the turtle's trace, colours painted to those figures.

3 Effects of Programming as Perceived by the Teacher

In this section, we describe the effect of programming education as felt by the teacher. The teacher's programming education experiences by now is as follows: LOGO language — 30 hours x 4 classes x 4 years (480 hours), Dolittle language — 20 hours x 4 classes x 4 years and 16 hours x 2 classes x 3 years (320 + 96 hours), approximately 1,200 students in total.

In programming classes, as students are enjoying the classes, they come early into the computer room and eagerly wait for the class to start. When the ending

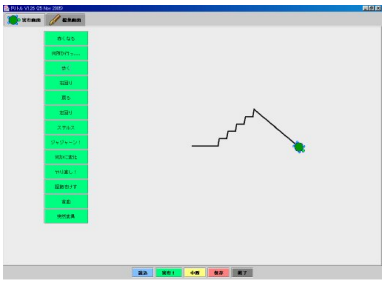

Fig. 6. Button program crafted by a student

bell rings, as students are really concentrating on their task, they are reluctant to leave their seat. Those attitudes are not seen so often in other subjects, and are very impressive to the teacher. From those (and many other) observations, we thought that programming classes should have positive effects on various aspect of the students, as listed below.

(1) logical/procedural thinking abilities and inference abilities

Program codes are executed in a line-by-line, step-by-step fashion. Therefore, to make programs, one always has to think in term of the orders of actions. This property of programming naturally enforce students to make reasoning on procedures. For example, many students initially do not care much about the order of descriptions in their code. However, they soon experience that the order of description leads to difference in resulting pictures, and start to think in step-by-step fashion.

In the same line, some students initially specify numeric parameters (such as movement distance or XY-coordinates) at random, but soon they have the desire to place figures at their will, and start to guess (or calculate) those numeric parameters to create desired pictures. As the result, those students start using their logical thinking and inference ability, and therefore, their logical thinking and inference abilities will become enhanced.

(2) concentration, observation, endurance

In most occasions, programs will not work as in the students' expectations. Especially at its initial experiences, students receive many errors from their programming environment. However, at the same time, they observe their colleagues, who are all similar in study performances, successfully run the code.

Therefore, they start to examine their code carefully, discover their flaws, correct them, and then the situation improves — positive feedbacks. Repetition of this process apparently enhances students' concentration on the subject, careful observation, and endurance when the result were not as they expected.

(3) problem solving abilities (logical thinking, imagination)

As the students does their trial-and-error on their programs, they have to solve many problems. Some problems can be solved by enhancing their code, but others might require them to adjust their plans for the resulting programs' outcome. In total, they need to adjust their code and goals simultaneously. Those tasks apparently enhance students' logical thinking over the problem, and also imagination as to what can be achieved through their programs.

One of the biggest advantage of the programming is that levels of the goals for each students are automatically adjusted according to their abilities. The key point is that each of the student determine their goal themselves; those who have advanced skills will set the advanced goals, while those with only basic skills will set the goal within their abilities.

The net result is that each student has the goal just appropriate for their abilities. Then they take the challenge over their goals and can develop their abilities steadily.

(4) understandings of the principles of software and computers

Our original (and still one of the main-) goal of programming education is to enhance students' understandings of the principles of software and computers, as in fast execution, large flexibility (almost everything can be changed through software), rigidness (a program does what is written in the code and nothing else, however meaningless as it seems).

For this kind of goals, programming classes have outstanding effects on students. In our observations, many students seems to become confident about what the computers are and how they operate.

4 Effect of Programming as Perceived by the Students

To asses the teacher's feelings listed above, we planned to obtain more objective data. Therefore, in the school year 2005 classes, we have conducted enquiries over the students. The number of students was 55 in total (sum of two classes), all in 3rd grade.

The enquiries were taken in two stages; the 1st enquiry (October 2005) consists of free description about the classes, and the 2nd enquiry (February 2006) consists of multiple choice questionnaire. This configuration was chosen because we could pick up interesting points from the free description and then examine them more thoroughly with quantitative data.

4.1 Free Description Enquiry

In the free description enquiry, the instruction given to the students were: "if you got something useful from your programming classes, describe them whatever they are." No further restriction on the answer was given, and the students have written their thought freely.

Table 1. The categories obtained from the free description

description	# of students
A. Understandings of mathematics	25
B. Thinking experiences	17
C. Understandings of computers	8
-. other kind of answers	14
"none"	2

Answer from the students could largely be classified in the categories A-C shown in the Table 1. Some representative answers are given in Fig.7.

A. I could understand XY-coordinates.
A. It was useful to figures in math.
B. I could think up of various ideas.
B. I could concentrate on a task for some duration.
C. I could understand how video games are made.
C. I could type the keyboard quickly.
-. I could see the mechanism of timer object a little.
-. It was interesting because I could make various figures.

Fig. 7. Some representative answers from the free description

It was a bit striking to us because students seems to learn more about mathematics and thinking abilities in general, rather than principles of computers, which were the original objectives of the classes.

However, learning in this ages are often intertwined; learning from one subject are frequently transferred to another. If a topic in a subject transfers to many other areas, it will be more worthwhile to teach that topic. Therefore, we thought this is one of the strong points of programming as a teaching material. So, we conducted the second enquiry to investigate the situation more thoroughly.

4.2 Multiple Choice Enquiry

As noted above, we crafted the multiple choice enquiry by picking up interesting (as we felt) points from the free description, made each of them as a statement (we tried to retain original words when possible), and let the students select from 4 choices (Yes, Perhaps, Perhaps Not, No).

In the followings, we show results and related discussion for each of the categories A-C. The percentages shown are the ratio of the students who have selected the positive choices (Yes or Perhaps).

A. Understandings of mathematics In Table 2, results of category A (math) questions are shown. From (1), we could see that most of the students agree that programming classes were useful for understanding of math XY-coordinates. From (2), approximately half of the students are feeling that figures in Dolittle is more comprehensive than figure in math classes. From (3), only one third of students seems to think that Dollitle classes were useful for understanding math functions.

Table 2. Result of multiple choice enquiry (Category A)

description	%positive
(1) I could understand math coordinates (X-coordinate, Y-coordinate, + and − direction of axis)	68.7%
(2) Turtle movements were more comprehensive than figure in math classes	41.2%
(3) It was useful for math linear function ($y = ax$) understandings	29.4%

With respect to (1), in Dolittle classes, students had to specify XY-coordinates on figure movements, animated movements, GUI button positions and so on. Therefore, students specified coordinate values in trial-and-error fashion. As the coordinate values are reflected to the positions on the screen on Dolittle, many students seem to understand their relations.

With respect to (2), the teacher has analysed the correlation with students' math performances. As the result, those students who are not good at math answered more positively in this question. In general, students not good at math have strong negative attitude toward math classes. Therefore, to those students, figures in Dollitle seems to be more acceptable than figures in math.

With respect to (3), the percentage of positive answers was not high. We guess the reason as follows: those students good at math already understand functions, so programming classes do not help, and those not good at math cannot find relation among math functions and graphics line segments.

However, correlation analysis against math performance shows that students not good at math have higher percentage of positive answers. Therefore, even though they cannot understand relation among functions and lines, they did used coordinates and slopes in their programs, and this might indirectly enhanced understandings of math linear functions.

In total, we might conclude that Dolitle classes have positive effects over students' math understandings, especially in the area of figures and linear functions for students not good at math.

B. Thinking experiences In Table 3, results of category B (thinking) questions are shown. In this category, many students (more than half) answered positively to all question (4)-(7); therefore they are feeling positive effects on

endurance, concentration, thinking abilities and imagination. Especially, percentage for thinking abilities (6) is strikingly high.

Table 3. Result of multiple choice enquiry (Category B)

description	%positive
(4) I could gain endurance and continue a task for long duration.	51.0%
(5) I could concentrate. I got an ability to concentrate on the task.	56.9%
(6) I got an ability to think.	68.6%
(7) I got more imagination than before. I got creativity.	52.9%

In the programming classes, when students are creating their programs, they stay still in front of the screen, modify their programs, execute them, modify again and the process continues. No sound other than key clicks are audible. Such situation lasts for hours, which is not at all imaginable in other subjects! From the above situation, it is no wonder that they feel their endurance (4) and concentration (5) were enhanced.

With respect to (6), students need to think in term of procedural steps and guess outcome of the code they are writing. Additionally, when writing programs, they have to think in term of large views, and need to skip useless part. Therefore, it is natural that students feel enhancement in their thinking abilities.

With respect to (7), we had some informal interviews to the students. Their answer in general was that, they have little chance of using their creativity in junior high-school classes (it is problematic side of Japanese school education, we admit). Maybe, creativity is required when imaging situations in the classes of literature, or free drawing on art classes, and some more, but not much anyway. In Dolittle programming classes, we encouraged students to write whatever drawing or animation program they would like to create, and students actively performed trial-and-error to complete their work. Therefore, it is natural that they feel their creativity were enhanced.

C. Understandings of computers In Table 4, results of category C (computers) questions are shown. In this category, many students (more than half) answered positively, thus many students feel that their understandings on computers are enhanced. Especially, (10) seems to indicate that students have grabbed an idea of "procedures automatically executed by computers."

Understanding principles of computers were original (initially-planned) goal of our programming classes. In the classes, students have created animation programs or interactive drawing program (controlled by GUI buttons). Creating those software themselves might help students to grab what the software actually is. For years we have been thinking that programming in classes is useful measure to teach principles of computers. The above result reconfirms our thought.

Table 4. Result of multiple choice enquiry (Category C)

description	%positive
(8) I could understand how computers are organised.	62.8%
(9) I could understand how video games are made.	64.7%
(10) I could understand general structure of programs and also "programs behave how when I modified them in which way."	74.5%

5 Discussions and Future Research

In our experiences, programming classes have many positive side effects (i.e. effect other than enhancing students' programming skills and understandings of computers). There are literatures on those topics such as [5], [4] and so on, but we felt our use of Dolittle educational object-oriented language have increased the overall effect. We analysed those effects, based on the teacher's experiences, in section 3. Additionally, our intuition was confirmed in subjective tests (enquiry) of section 4.

We are willing to examine the theme further. For example, it may be more plausible to focus on students' changes in consciousness and thinking style. For example, if a student learn "to use his brain" through programming classes, he will continue to do so on other subjects and his performance will raise in general. "Computer sciene unplugged"[6] might have positive effect in this direction, and we would like to investigate collaboration of this scheme and our Dolittle scheme.

In general, as we believe, important aspects of education at junior high school ages are that a student have various chances to find something by himself, experience something new, and so on. Those findings and experiences might make him a good "learner," and thus make his life wonderful and plausible one.

References

1. Dolittle Programming Language. http://dolittle.eplang.jp/
2. Susumu Kanemune, Yasushi Kuno. Dolittle: an object-oriented language for K12 education. EuroLogo2005, Warsaw, Poland, pp.144-153, 2005.
3. Susumu Kanemune, Takako Nakatani, Rie Mitarai, Shingo Fukui, and Yasushi Kuno. Dolittle — Experiences in Teaching Programming at K12 Schools. The Second International Conference on Creating, Connecting and Collaborating through Computing (C5), IEEE, pp.177-184, Kyoto, Japan, 2004.
4. Richard E. Mayer, Jennifer L. Dyck, William Vilberg, Learning to Program and Learning to Think: What's the Connection?, Communications of the ACM, vol. 29, no. 7, pp. 605-610, 1986.
5. Seymour Papert, Mindstorms, BasicBooks, 1980.
6. Tim Bell, Ian H. Witten, Mike Fellows, Comoputer Science Unplugged – An enrichment and extension programme for primary-aged children, 2005. `http://csunplugged.com/`