Development of State-Based Squeak and an Examination of Its Effect on Robot Programming Education

Hiroyuki Aoki[1], JaMee Kim[5], Yukio Idosaka[2], Toshiyuki Kamada[3], Susumu Kanemune[4], and WonGyu Lee[5]

[1] Department of Computer Science Education, Graduate School, Korea University,
Anam-dong Seongbuk –gu, Seoul 136-701 - Republic of Korea
[2] Iitakahigashi Junior High School, Matsusaka City
927 Miyamae, Iitaka-cho, Matsusaka-shi, Mie, 515-1502 - Japan
[3] Department of Technology Education, Aichi University of Education
1 Hirosawa, Igaya-cho, Kariya, Aichi, 448-8542 - Japan
[4] Faculty of Biomedical Engineering, Osaka Electro-Communication University
18-8 Hatsucho, Neyagawa-shi, Osaka, 572-8530 - Japan
[5] Department of Computer Science Education, Korea University,
Anam-dong Seongbuk –gu, Seoul 136-701 - Republic of Korea
[e-mail: hiroyuki.aoki@inc.korea.ac.kr, idosaka@gmail.com, tkamada@acm.org, kanemune@acm.org, {jamee.kim, lee}@inc.korea.ac.kr]
*Corresponding author: WonGyu Lee

Received April 13, 2012; revised July 31, 2012; revised October 22, 2012; accepted November 19, 2012;published November 30, 2012

Abstract

Robot programming often sparks students' interest in programming, but it is not easy for them to program both procedure and reactivity of robot movements that are essential requirements. In this study, we reviewed in detail a new programming language, *State-Based Squeak*. It allows novice students to implement both procedure and reactivity of robots easily. The effect of this new language on robot programming education was also examined using a group of 28 middle school students. According to the results of analyzing the students' understanding of programming, reading and programming abilities the group that used State-Based Squeak (the experimental group) showed a higher completion ratio than the other (control) group. The significance of this study is that a robot programming language has been developed that addresses the concepts of both procedure and reactivity in such a way that middle school students can more easily learn how to program robots, something that is often difficult to attempt even for professional programmers.

Keywords: Robot programming, programming paradigm, state control

A preliminary version of this paper appeared in ICONI 2011, Dec. 15-19, Sepan, Malaysia. This version includes the result of the examination measuring effect of the State-Based Squeak using middle school students.

http://dx.doi.org/10.3837/tiis.2012.10.008

1. Introduction

Programming education helps students enhance their problem solving abilities [1] and develop higher-order thinking [2]. Therefore, programming is taught not only in university as a major field but also in middle school [3][4]. In Korean "Revised Curriculum 2007", programming was introduced into the Korean middle school informatics curriculum. Japan has also included programming into their curriculum as a mandatory course. The programming education included in Japanese middle school curriculum is primarily aimed at teaching the principles of measurement and control systems using a computer [5][6].

Robot programming is one of the most popular topics among various types of educational programming activities because robots attract students' interest and curiosity [7]. However, programming is not easy for novice students because the programming language and environment tend to be designed for professionals [8]. In order to implement a robot program, students need to consider thoroughly two key aspects, namely procedure and reactivity [9]. That is, procedure of the robot's preplanned behavior and reactivity to the events that the robot meets with must be considered for mobilizing the robot. However, the procedural language and reactive language are in two different paradigms, so it has been difficult to consider these two features simultaneously in a programming language for robots [10][11][12]. It is hard for novice programmers to describe reactivity in the procedural language, and conversely, to describe the procedures in the reactive language [13].

This study developed State-based programming language that can specify the procedure and reactivity based on Squeak, which is a programming language for novices. The effect of this new programming language on educational robot activities is also examined.

Chapter 2 will review the characteristics of programming languages required for educational robot activities. Chapter 3 will explain the developed state-based programming language and compare its programs with ones in the other paradigms. Chapter 4 will describe the method for examining the effect of the program developed in this study. Finally, Chapter 5 will discuss the effect the program had upon the middle school students used in the study.

2. The Rationale of the Developed Robotics Specific Features

This section deals with the features of programming languages that are required for educational robot activities. Additionally, the requirements for developing robot programming languages for novice students are also defined.

2.1 Procedure and Reactivity of Robot programming

Educational robot activities such as robot competitions augment students' motivation and creative thinking [7][14][15]. Since programming activities in general entail the use of abstractions such as numbers, letters, signs and objects, students experience difficulty in manipulating or designing programs. However, because robot programming involves the use of physical metaphors, students can plan robot behaviors by turning themselves into robots [3][4][16]. Programming is a cognitive activity that requires the creation of problem solving strategies [17]. Programming a system that implements planned behaviors of a robot is an activity that greatly contributes to the development and improvement of problem solving abilities [18].

Robots are programmed to be mobilized in unpredictable physical world [19][20]. A mobile robot not depends on an ideal environment that allows it to move about as programmed, but reactivity must also be considered to make the robot react appropriately to the various situations it may face because the physical space is subject to change [12][15]. Robot reactivity refers to the robot's ability to identify the situation of its surrounding environment using its sensors and to select the appropriate behavior. Accordingly, reactive programming entails setting the rules and protocols that determine the responses.

There are two key aspects in robot programming that must be addressed: "procedure" and "reactivity" [9]. Its procedure for realizing a robot's strategy and its reactions to events must all be considered. However, students' robot programs depended mostly on procedure and had difficulty in implementing reactivity [10][11]. Procedure is easily described in the procedural languages and thus is a different paradigm from reactive languages that is described in declarative style [12]. Therefore, it is hard for novices to describe reactivity in a procedural language, and describing the procedure in a reactive language is also difficult [13]. There have been plenty of studies done on consistency of them, but proper solutions have not been presented [13]. Only a few studies have been conducted on educational robot activities for novice programmers.

2.2 Approaches of Consistency with Procedural Features and Reactivity

Biggs and MacDonald (2006) suggested five approaches to programming robot reactivity which were event loops, reactive languages and behavior architecture, multi-threading, reactive functional languages, and hybrid architecture [9]. Among these approaches, event loops, reactive languages and behavior architecture are the most applicable to introductory activities. Details in these approaches and applicabilities are described in the following.

2.2.1 Event loop

An 'event loop' is used to realize reactivity in procedural programming [13]. Procedural programming is commonly used [21], including educational robot programming languages such as RoboLab [22] and NXT-G [23]. Procedural programming consists of sequence of commands for accomplishing a specific aim. If novices can read and write commands, they can construct and implement a program [24].

An event loop consists of conditional statements using sensors or timers to detect events and execute a response. Following is an example of an event loop in language *C* that triggers a beep when the input value of sensor 'A' goes down:

```
while (TRUE) {
   if (sensorA () < 100) { beep (); }
}
```

It is easy to denote an event loop because of its simple structure as shown above, but it is essential to consider the consistency of events that may occur. For example, the following contradictions typically occur:

First, novice programmers frequently forget to use the loop that is used to keep watching events [25]. They tend to concentrate more on the entire flow despite the fact that the events that may occur should be considered.

Second, reading the entire flow may fail if event loops are concentrated. If too many events are considered, the program structure will become overly complicated. In order to not lose the flow of a program with a complicated structure, the program is divided into modules, or

comments are written. However, dividing or commenting is very difficult for novice programmers because the criteria for division or comments are not clear [26]. Another weakness is the difficult maintenance/repair and expansion [13].

If following computer processing is difficult, a good way to proceed is to materialize the programming language in the way people think. Cricket Logo is such an example. Cricket Logo uses a control structure for a 'when' sentence that is executed whenever specified requirements are met [27]. Below is an example showing the "execution of the command in [ ] when the value of sensor 'A' is lower than 100."

```
when (sensorA< 100) [beep]
```

The control structure beginning with 'when' is an implicit event loop that detects an event.

LogoBlocks [28] and RCX Code [29] are easy for young students to deal with because they use graphical expression. However, those are not suitable to change the reactive rules in response to progression of the work. For this reason, FlogoII [30] controlled reactivity by making the distinction between 'steps' that are executed sequentially and 'processes' that are an implicit event loop. However, a sufficient understanding of the two models precedes the use of 'steps' and 'processes' in programming.

2.2.2 Reactive Languages And Behavior Architecture

Programming that considers reactivity makes diverse interactions possible, so the range of its application is expanding. Event-driven programming, which has been generalized with the development of desktop applications, is a type of reactive programming. Because reactive programming language automates event loop processing, it must focus on the rules.

More complicated rules are required for reactive robot programming than for desktop applications [9]. In robot programming, a rule-based language is provided suitable for the target robot, e.g. LegoSheet [31], Altaira [12] or RobotWorks [32]. The rules are expressed in a table, but there is also a data flow language described in the format of sensor input and motor output like that of the diagrammatical representation of an electric circuit. A rule-based language has the convenience of automation but also difficulty understanding the principles for program operation. The rules should be mixed with other content for a sequential process. When contractions occur as rules increase or when priority should be determined, rule management must be specified and high-level logical thinking is required [13].

Etoys, which is able to simulate multiple objects, can express reactivity well [1]. Etoys can be used in the desktop programming environment but is also useful for teaching robot control programming in schools as with Squeakbot [33] or Physical Etoys [34]. Script that is one of the program units of Etoys plays the role of 'event loop' that realizes reactivity through interaction with the system. That is, reactivity is realized by judging events within the script and by describing the contents that can be responded to. Etoys is easy to understand the contents of the script because it is a procedural program. However because the script of Etoys has two roles of being executed singly and of event loop being called repeatedly, it is difficult to use them properly [35].

2.3 Requirements for Novice-oriented Procedural and Reactive Features

As seen above, novice student programmers need to acquire "procedural programming" and "reactivity" together even for educational activities associated with robotics. This study specified the following requirements for students writing programs based on previous studies:

First, procedural programming whose principles are easy to understand is necessary. A procedural language that reveals the procedures for computer processing is more suitable for a learner so that she/he may better understand computer operational principles to write programs. For the aspect of reactivity, a complicated structure should be eliminated by simplifying the program with event loops.

Second, an appropriate standard is necessary for breaking up the program into units. Repetitive behaviors are structured by the called procedure in robot programming so that all behaviors within the program must define the procedure. Defining the procedure and setting names for calling them can clarify the meaning of the procedure and also improve the functionality of the program. The program should be broken up into a series of sets to define each procedure, and the programmer must set the definite standard for breaking up the program into sets. The division of a program into partitions results in testing each partition, and in teaching robotics, program partitioning and testing allows students to share ideas and improve their teamwork skills [14][15].

Third, it is necessary to visualize changes in the reactive rule. It is sometimes required to express reactivity during a 'procedure' that is called repeatedly. This situation refers to the 'state' in the behavior architecture so that the 'repetition of a procedure' can be explained as a concept of the 'state.' Reactivity can be divided into two rules. First, there is a 'temporary reaction' that goes back to the original behavior as the specific situation ends. A temporary reaction means the change in behavior within the repetition of a procedure because it returns to the original behavior. Second, there is a 'permanent reaction' that does not return to the previous situation but instead moves to the next behavior. A permanent reaction does not go back to the previous state, so the reaction itself is a change and it also changes the procedure of being continuously called by using the control command. This refers to state transition. The change in reactivity needs to be visualized based on the finite automation model that divides the system into finite states.

Fourth, a 'state' mechanism is necessary. A 'state' mechanism looks similar to a repetition structure in that it is continuously implemented. Accordingly, students might confuse the state concept with the repetition structure concept. For this reason, a 'state' mechanism is adopted, and the event loop is used for expressing the parts that overlap between the concepts. A state procedure is created and used for handling iterations instead of using a repeat command. A program can be broken up into a series of behaviors by using state procedures as mentioned before.

3. Development of State-Based Squeak to Support Robotics Programming

While writing robotics programs, students experience quite a bit of difficulty with conventional *procedural programming* because of its weakness in handling or controlling *reactivity* that is a basic functional requirement for robotics control [9][12].

For reactive control, *rule-based programming* has an advantage over procedural programming [12]. Rule-based programming is used extensively in industries to handle automation and electromechanical processes, such as automatic doors, elevators and amusement park rides where reactivity is a fatal issue. A rule-based program is described as a set of simultaneous rules consisting of the relationships between the inputs of sensors and outputs of actuators. For example, the *Ladder Diagram (*LD), which is a typical rule-based programming language, is based on the electrical circuit model. The program's similarity to

diagrammatical representation of an electrical circuit does not require of programmers any abstract concept of procedural programming languages [36]. This rule-based principle is suitable for dealing with the reactivity encountered in the real world, and the static characteristics of the rules are suitable to online execution visualization of control equipment. So, whereas *rule-based programming* has the advantage of handling reactivity, its ability to handle complex sequences tends to be complicated [36].

Controls for robotics require both reactivity and sequential performance in their programming. Therefore we adopted the principle of *state-based programming* that integrates procedural and rule-based programming paradigms. We materialized this idea with a new programming language called *State-Based Squeak*.

3.1 The Principle of State-Based Programming

The concept of the thinking behavior of a program with *states* can be seen in a *finite state diagram* (FSD) and state-based control [37]. The LD is also extended for its sequential ability with a *Sequential Function Chart* using the state concept [38].

The principle of state-based programming in this study is to define an *event loop* as a fundamental unit of a program and form it into a named procedure like a *function* in programming language *C*. The event loop is the fundamental mechanism of reactivity for controlling robots and can be seen in both procedural and rule-based programs. A state-based program consists of event loops. We consider each event loop as one of the *states* of a computer system because it is fundamentally in a state of waiting for some event to occur and the same reactivity is performed.

We developed the programming language, *State-based Squeak,* a derivative of *Squeak* [39], to implement the concept of state-based programming. **Figure 1** shows a sample program, which is for *an electric fan* that is controlled in alternate action by clapping sound:

```
stateStandBy
    motor1 off.
    (self detectClap) ifTrue: [ self transitTo: #stateWorking ].

stateWorking
    motor1 on.
    (self detectClap) ifTrue: [ self transitTo: #stateStandBy ].
    (self frontSensor > 300) ifTrue: [ self beep ].
```

Fig. 1. An example program of State-Based Squeak

Because the fan can start or stop by the same clapping sounds it is considered that the fan requires two different states: ***stateStandBy*** and ***stateWorking***. By defining reactivity in each procedure, the application framework forms an event loop by continuous calling the user defined procedure and performs the specified reactivity of the current state. At first, assume that stateStandBy is active. When **transitTo:** method is called in the certain condition, the current state is changed to specified state (stateWorking). This action is a *state transition* that performs sequential control. Now reactivity is changed and motor1 getting rotated and if someone got closed to the fan, the buzzer warned one's approach**.**

3.2 Comparing Three Types of Programs

In order to better characterize a state-based program, we compared rule-based, procedural, and state-based programming (**Fig. 2**, **Fig. 3**, **Fig. 4**, **Fig. 5**). In this chart, each program is

displayed with a typical diagram to show its structure. Accordingly, the pairs of flowchart and Interactive C [40] for procedural programming, LD and Structured Text (ST) [38] for rule-based programming, and FSD and our State-Based Squeak for state-based programming were adopted. The following were used to compare the programs:

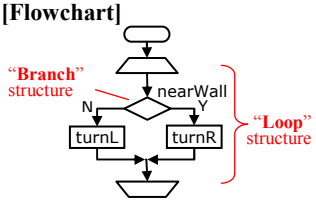
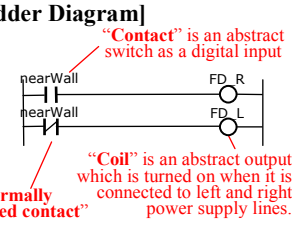
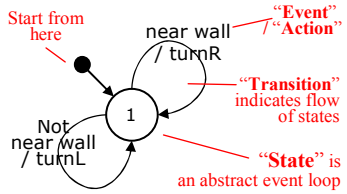
(1) Reactive control: a mobile robot running parallel to a side wall (**Fig. 2**).

(2) Sequential control: switch 's1' starts a beep and switch 's2' stops it (**Fig. 3**).

(3) Timed sequential control: traffic lights change alternately between green (GO) and red (STOP) every 5 seconds (**Fig. 4**).

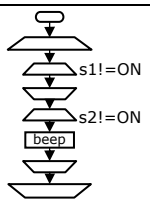
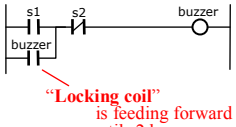
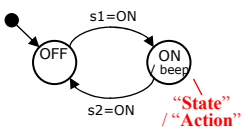
(4) Control of robotics competition: a mobile robot pushes balls from one side of a playing field to the other on a surface on which color is constantly changing (**Fig. 5**).

The first three represent basic components of control and the fourth is a practical application. Robotics competitions (4) originated with the *Robo-Pong Contest* that was first held at the annual MIT Robot Design Competition in 1991 [40]. The algorithms adopted in the programs shown are simplified from the original due to space constraints.

Procedural programming	Rule-based programming	State-based programming
[Flowchart]  	**[Ladder Diagram]**  	**[Finite state diagram]** 

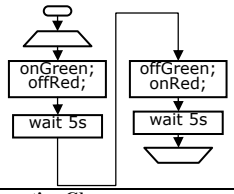
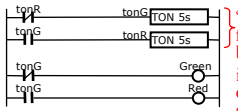
[Interactive C]
void main() {
while(1) {
if(analog(DISTANCE)<NEAR)
{
fd(RIGHT); off(LEFT); }
else {
fd(LEFT); off(RIGHT); }
}
}
	[Structured Text]
	Q_FD_R := I_NEARWALL;
	Q_FD_L := NOT I_NEARWALL;
	
	(* The prefixes Q_ and I_ are assigned to outputs of actuator and inputs of sensor respectively for readability. *)
		[State-based Squeak]
		state1
		(distance value<=NEAR)
		ifTrue:[
		leftMotor fd. rightMotor off]
		ifFalse:[
		leftMotor off. rightMotor fd]

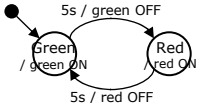
Fig. 2. Reactive control programs of three programming paradigms

Procedural Programming	Rule-Based Programming	State-Based Programming
[Flowchart]  	**[Ladder Diagram]**  	**[Finite state diagram]** 

[Interactive C]
void main() {
while(1) {
while(digital(S1)!=ON);
while(digital(S2)!=ON) {
beep(); }
}
}
	[Structured Text]
	IF I_S2 THEN
	Q_BUZZER:=FALSE;
	ELSEIF I_S1 THEN
	Q_BUZZER:=TRUE;
	END_IF;
		[State-based Squeak]
		stateOFF
		(s1 on) ifTrue:[
		self transitTo:#stateON].
		stateON
		(s2 on) ifTrue:[
		self transitTo:#stateOFF].
		buzzer beep.

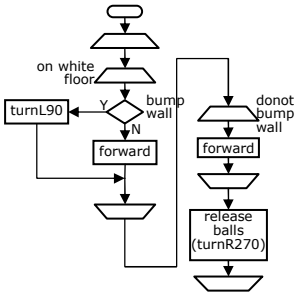
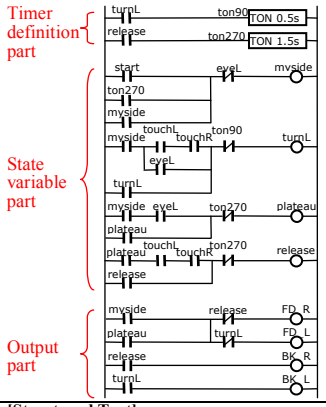
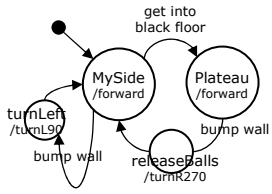
Fig. 3. Sequential control programs of three programming paradigms

<table>
<tr><th>Procedural Programming</th><th>Rule-Based Programming</th><th>State-Based Programming</th></tr>
<tr><td>[Flowchart]
 
</td><td>[Ladder Diagram]
 

"Timer function block" is turned on after the specified time during input is on</td><td>[Finite state diagram]
 
</td></tr>
<tr><td>[Interactive C]
void main() {
  while(1) {
    on(GREEN); off(RED);
    sleep(5.0);
    off(GREEN); on(RED);
    sleep(5.0);
  }
}</td><td>[Structured Text]
TON_G(
  IN:=NOT TON_R.Q, PT:=TIME#5S);
TON_R(
  IN:=TON_G.Q, PT:=TIME#5S);
(*These define timer functions with parameters of a start trigger of IN and an ignition time PT)

Q_GREEN:= NOT TON_G.Q;
Q_RED:=TON_G.Q;</td><td>[State-based Squeak]
stateGreen
  ledGreen setON.
  (elapsed >=5) ifTrue:[
    ledGreen setOFF.
    self transitTo:#stateRED]
stateRed
  ledRed out: 100.
  (elapsed >=5) ifTrue:[
    ledRed setOFF.
    self transitTo:#stateGREEN]</td></tr>
</table>

Fig. 4. Timed sequential control programs of three programming paradigms

<table>
<tr><th>Procedural Programming</th><th>Rule-Based Programming</th><th>State-Based Programming</th></tr>
<tr><td>[Flowchart]
 
</td><td>[Ladder Diagram]
 
</td><td>[Finite state diagram]
 
</td></tr>
<tr><td>[Interactive C]
void main() {
  while(1) {

    while(!digital(EYE_L)) {
      if(digital(BUMP_L)
        && digital(BUMP_R)) {
        bk(LEFT_MOTOR);
fd(RIGHT_MOTOR);
        sleep(0.5);
      } else {
        fd(LEFT_MOTOR);
fd(RIGHT_MOTOR);
      }
    }

    while(!(digital(BUMP_L)
      &&digital(BUMP_R))) {

  fd(LEFT_MOTOR);fd(RIGHT_MOTOR)
;
    }

    releaseBalls();
  }
}

void releaseBalls() {
    fd(LEFT_MOTOR);
bk(RIGHT_MOTOR);
    sleep(1.5);
}</td><td>[Structured Text]
(* Timer definition part *)
TON90(IN:=TURNL, PT:=TIME#0.5S);
TON270(IN:=RELEASE,
PT:=TIME#1.5S);
(* State variable part *)
IF NOT I_EYEL THEN
  MYSIDE:=FALSE;
ELSEIF I_START OR (PLATEAU AND TON270.Q) THEN
  MYSIDE:=TRUE;
ENDIF;
IF TON90.Q THEN
  TURNL:=FALSE;
ELSEIF MYSIDE AND ((I_BUMPL AND I_BUMPR) OR I_EYEL) THEN
  TURNL:=TRUE;
ENDIF;
IF TON270.Q THEN
  PLATEAU=FALSE;
ELSEIF MYSIDE AND I_EYEL THEN
  PLATEAU=TRUE;
ENDIF;
IF TON270.Q THEN
  RELEASE:=FALSE;
ELSEIF PLATEAU AND I_BUMPL AND I_BUMPR THEN
  RELEASE:=TRUE;
ENDIF;
(* Output part *)
Q_FD_R:=(MYSIDE OR PLATEAU) AND NOT RELEASE;
Q_BK_R:= RELEASE;
Q_FD_L:=(MYSIDE OR PLATEAU) AND NOT TURNL;
Q_BK_L:= TURNL;</td><td>[State-Based Squeak]
stateMySide
  (leftEye on) ifTrue:[
    self transitTo: #statePlateau].
  (bumpLeft on and: bumpRight on) ifTrue:[
    self transitTo: #stateTurnLeft].
  leftMotor fd. rightMotor fd.

statePlateau
  (bumpLeft on and: bumpRight on) ifTrue:[
    self transitTo: #stateReleaseBalls].
  leftMotor fd. rightMotor fd.

stateTurnLeft
  leftMotor bk. right_motor fd.
  (elapsed>=0.5) ifTrue: [
    self transitTo:#stateMySide].

stateReleaseBalls
  leftMotor fd. rightMotor bk.
  (elapsed>=1.5) ifTrue: [
    self transitTo:#stateMySide].</td></tr>
</table>

Fig. 5. Practical robotics programs of three programming paradigms

We analyzed the advantages and disadvantages for each programming paradigm shown in **Table 1**. The following factors were selected considering the key issues confronted by novices: (1) intuitiveness: A factor for the creating and reading. Its parameters are requiring less difficulty, less base knowledge; (2) traceability: A factor for problem solving (debugging). Its parameters are understandability of the exhaustive reactivity for each situation and the flow of processes; and (3) extendibility: A factor for improving. Its parameters are eases of handling complication and extension of the functions.

Table 1. Advantages and disadvantages of three programming paradigms

Perspective	Procedural programming	Rule-based programming	State-based programming
Intuitiveness	**Advantages:** Simple timed sequential control is described in a straightforward manner. **Disadvantages:** Code structures tend to be nested and complex. The structured programming restriction causes inconvenience. Any expressions that indicate the meaning of each part are not guaranteed.	**Advantages:** Simple reactive control is intuitive. The name of each state variable indicates the meaning of its code. **Disadvantages:** Sequential controls are complex with many steps and constructed differently according to the sensor inputs or elapsed time.	**Advantages:** Generally accustomed procedural expression is available. A states' name indicates the meaning of the each code and progress of execution. Sequential controls are constructed in the same form, irrespective of using sensor inputs or elapsed time.
Traceability	**Disadvantages:** The unstable active part in the code makes it difficult to trace unexpected executions.	**Advantages:** Static rules and wide coverage are good for output verification and runtime tracing.	**Advantages:** Action and reactivity displayed in units of states ease execution traceability.
Extendibility	**Disadvantages:** Modularization is based on vague criteria and not promoted. Nested structures make it difficult to reconstruct the program.	**Disadvantages:** Appending functions or any change require extensive investigation because the rules have complex relationships.	**Advantages:** Program code is guaranteed to be divided into small units by states, which correspond to the FSD design.

The advantages of the proposed state-based language are as follows: state-based language ejects iteration for event loops from a program and allows a simpler structure consisting of less nesting and more concise structures compared to procedural language. State-based language is concise in terms of procedural notation and division in states compared to rule-based languages. The motion and reactivity of robots are displayed clearly and definitely using the unit of states. The advantages of State-Based Squeak for robot programming at the novice level compare favorably with conventional programming concepts. Consequently, it was determined that the state-based program's closeness to design using a finite state diagram resulted in easy construction, debugging and extension of the program.

4. Experiment Study

This study examined the effect that a state-based programming language had on the ability of novice programmers' to describe a robot's procedural and reactive behavior. The experiment was conducted with two groups that had equal programming environments but different approaches. The experimental group was given the state-based programming environment while the control group was given a conventional procedural programming environment. Both

the experimental and control groups received 7 lessons which were taught by the same teacher. The findings of the experiment are described below.

4.1 Participants

First, the participating students were in the 9th grade and had already acquired basic "programming skills and tips on programming" before they participated in the study. That is, they learned sequencing and repetition in procedural programming through an educational programming language. The students were divided into two groups, an experimental group (7 boys, 7 girls) and a control group (6 boys, 8 girls). They were homogenously grouped based on their combined academic achievement scores in the previous grade (Wilcoxon rank test, p-value>0.1). Second, the teacher who had been teaching technology for 26 years was in charge of the experiment. In addition, he had 10 years experience of teaching programming and 2 years experience of teaching robot programming. He provided a minimum of explanation on the programming language for the students to learn and write state-based programs on their own. The lesson plan was student-centered, meaning the teacher was least involved in the class. His responsibility was mainly to lead the students to go through each of the stages of the lesson plan, check if the students were doing what they had to do, and respond to problems as they occurred.

4.2 Materials

The materials were specially prepared and designed for the experiment.First, the topic of robot activity was 'Computer Programs and Measurement/Control Systems' listed in the Technology and Home Economics course curriculum of the middle school. Accordingly, the learning objective corresponded to the course objective, "simple measurement/control systems using programs" [5]. Second, students used desktop PCs with Microsoft Windows XP for robot programming, and a robot manufactured by Studio MYU (**Fig. 6**).

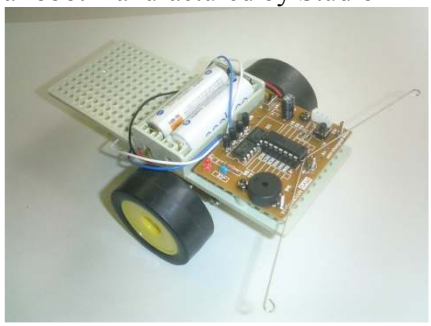


Fig. 6. "MYU Robot" by Studio MYU

In this examination, we used a visual programming environment of Squeak, namely *Rtoys*, to program MYU Robots. There are a total of eight move commands available for the MYU Robot, but only four were used in the Rtoys ('Forward,' 'Reverse,' 'Right Turn' and 'Left Turn'). The MYU Robot could make a curve with the combination of these primitive commands. Third, teaching materials were specially developed to teach the contents of the State-Based Squeak and distributed as handouts in every class. Each class lesson was composed of three parts. In the first part, an instruction was provided about the outline of the problem to be solved by students with what they learned in the class. In the second part, students did exercises to understand concretely what they had learned. In the third part, assignments were given to check if the students understood what they learned. Students were

asked to solve the assigned tasks on their own. All subjects challenged the students to complete the tasks assigned to them on their own. The contents of the teaching materials and the algorithm used for solving the tasks were the same for the two groups. However, the control group used repeat structures and escaping repetition command "exit loop" according to the procedural programming system, while the experimental group used state scripts and the state transition command "transit to" according to the State-Based Squeak.

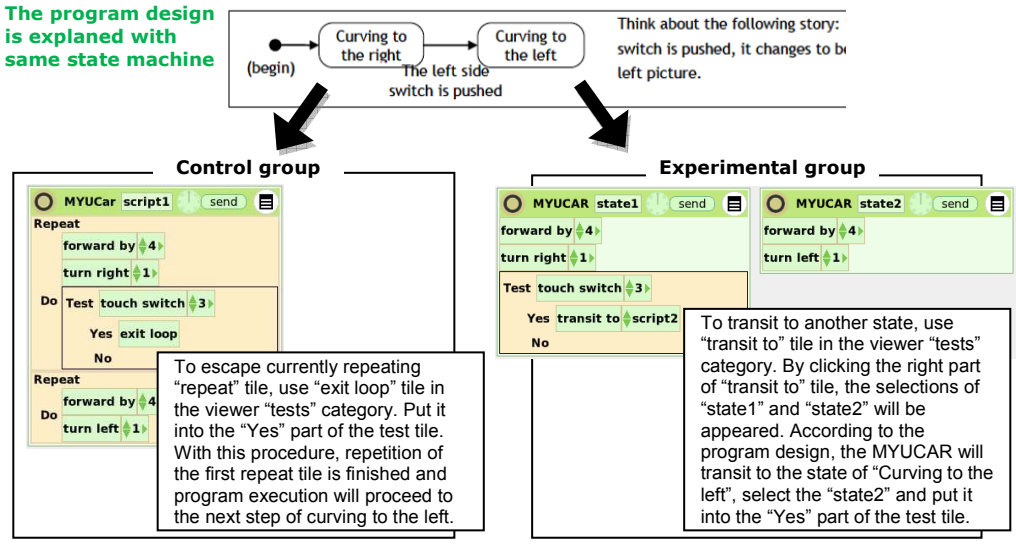


Fig. 7. Two types of teaching materials (translated)

Fig. 7 shows part of the explanation on "Change the two repeated behaviors" used in the 7th class.

4.3 Procedure

Table 2. Contents of experiment class

Class#	Step	Concept	Contents	#	Exercise assignments
1	Intro.	Sequencing	Basic manipulation of tile-based language	1	Self-programmed animation
2	Learning	Repetitive actions	Drawing figures – repetitive behaviors	2	Draw polygonal shapes such as triangles and squares.
3-4		Temporary reaction	Collision avoidance (turn) – use of sensors	3 4	Turn left/right to avoid obstacles.
5		Permanent reaction	Collision (reverse) – self-solution of change in repetition	5	Regress at the time of collision.
6-7			Command for sensor activity – Practice changing repetation	6 7	Move at the first command and stop at the second command. Assign the commands given in the 6th class to reverse right and left using tactile sensors.
8	Evaluation	Writing of a program	Self-programmed motion	8	Make self-programmed motions (Write a program that implements many behaviors)
9		Testing	Paper test		Answer 10 questions of varying degree of difficulty
10	Review		Recall of the programming process		Recall difficulties encountered and interesting things

A total of 10 classes were offered, one class per week. For fair comparison, the order of receiving the class changed on a weekly-basis, that is, the class was given first to the control group in one week and then to the experimental group in the next. Each class lasted 50 minutes in the following order: 1) the first 20 minutes were used for the teacher's instruction and students' exercises; 2) the next 25 minutes for students to complete assignments for assessing their understanding of what they had learned; and 3) the final 5 minutes for writing self-evaluation results on a class card. **Table 2** shows the class procedure and contents covered throughout the 10 classes.

4.4 Measurement and Instruments

This study conducted an evaluation consisting of three steps to examine the effect of the state-based programming system developed in this study.

First, the learning achievement was evaluated for each class. In every class the content to be taught was introduced to the students and then their understanding evaluated. The evaluation focused on the degree of students' understanding on the lecture which was about programming concept.

Second, the self-programmed system was evaluated. The objective of the assignment was to 'Make a program that realizes as many behaviors as possible using tactile sensors.' This assignment was aim to encourage students to make their programming system complicated. A list of commands was given to students to help them realize as many behaviors as possible, and then students were asked to mark a check for the commands they used. The complexity of the program was evaluated using the value of M, the Cyclomatic Complexity (CC) of McCabe (1976) [41]. The value of M was computed by drawing a control flow chart and counting the number of nodes (N), the number of edges (E) and the number of connected components (P). It is denoted by the following formula:

$$M = E - N + 2P \tag{1}$$

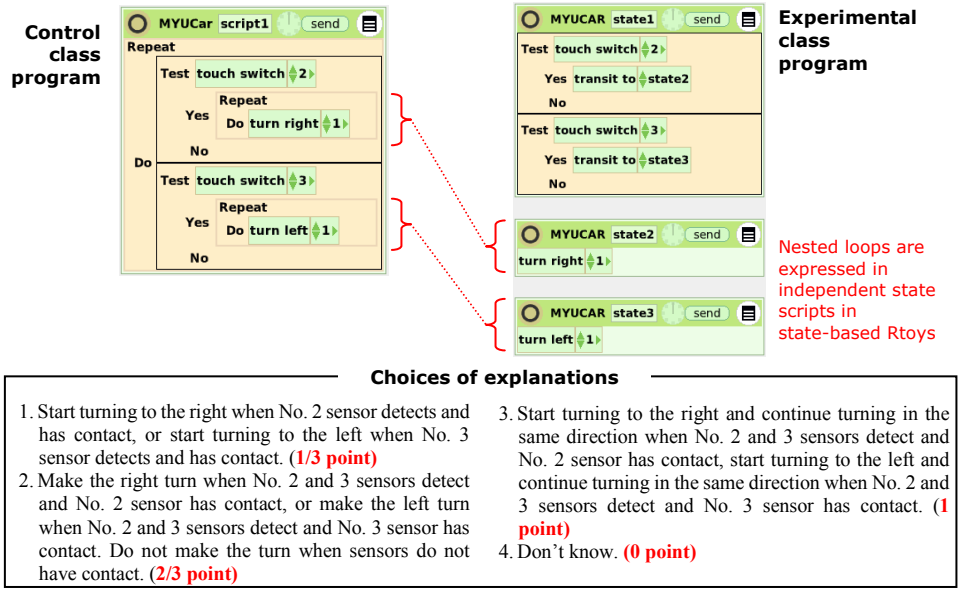


Fig. 8. Two types of programs for same problem assignments (translated)

Third, the program readability test had 10 question items with varying degrees of difficulty. An example question is shown in **Fig. 8**. The program included 3 repeats/states. In the control group's program, the second and third repeats are nested in the first repeat. In the experimental group's program, all in the script are parallel. Students were asked to give correct answers for the questions. **Table 3** shows the formulas that are the rationale for grading the questions. Students' answers were graded based on the problem solving process as shown in **Table 3**.

Table 3. Structures of programs and choices

#	**Presented program** (= 1 point answer)	**2/3 point answer**	**1/3 point answer**
Q1	(A–B–)+	(AB) +	A–B–$
Q2	(ε+ (sA)+) +	(ε+ (sAε)+)+	ε+ sA+
Q3	(A+{sB | tC})+	A+{sB | tC}+	A+sB+tC+
Q4	(A+sB–)+	(A+sB)+	A+sB–$
Q5*	ε+sA+	(ε+sA) +	ε+sA+t$
Q6	A+sB+	(A+sB+t)+	(AB) +sB+
Q7*	(ε+{ sA+ | tB+})+	(ε+{sA+ | tB+}t)+	ε+sA+tB+
Q8	ε+sA+t$	(ε+sA+t)+	A+{ s | t }$
Q9	A+sB+t$	(A+sB+t) +	(AB)+sB+t$
Q10	(ε+{ (sAB)+ | (tAC)+ })+	**(For control group)** (ε+(sAB)+ε+(tAC)+)+ **(For experimental group)** ε+{s | t}A(sB)+(tC)+$	**(For control group)** ε+{s | t}A(sB)+(tC)+$ **(For experimental group)** (ε+(sAB)+ε+(tAC)+)+

* marked questions include nested loops for the control group

Signs are implemented in order from the left. The meaning of each sign:
A, B, C: Robot behavior
ε: No motion (stopped)
$: End of program
–: The previous behavior continues over 0.5 second
(the period has a special meaning such as 90-degree turn).
+: The previous behavior is repeated.
s,t: Event occurrence (sensor contact)
( ) : Bundle of behaviors (used for repeating a series of behaviors with a "+".)
{ | } : Selective behavior of plural expressions divided with a vertical line.

Q10 was constructed to make the control and experimental groups answer in different ways. The experimental group was asked to solve the problem by changing the state, while the control group was asked to solve the problem through the nest. For instance, Q5 and Q6 are similar in structure, but Q5 requires programming by nesting the repeat, while Q6 asks for programming by avoiding a repeat. On the other hand, the control group used a method that changed all the states of two questions, showing no difference in the way of solving the problem between two questions. The problem solving method for Q5 and Q6 are applied equally to Q8 and Q9.

5. Results

To measure the effect of state-based programming, this study first compared the completion ratio of exercises between the two groups based on the program concepts and then analyzed the free programming assignments that allowed students to practice as many robot behaviors as they could, and then, finally, a program readability test was conducted to evaluate the

overall understanding the students had of the program. The analyses results are provided in the following sections.

5.1 Completion Ratio of Exercises

Table 4 shows the completion ratio for each assignment based on the students' understanding of the program concept. A two-tailed Fisher's exact test was used to statistically analyze the difference between the control and experimental groups.

Table 4. Completed examinee ratios for each assignment

Concept	Group	Completed examinee ratio	P value
Repetition	Control	0.63	.058
	Experimental	0.75	
Temporary reaction	Control	0.52	.047*
	Experimental	0.69	
Permanent reaction	Control	0.17	.007**
	Experimental	0.44	

Significant level: * p<.05, ** p<.01

According to the results, the two groups showed a difference in repetition of 12%, which is not statistically significant. However, for the completion ratio for temporary reaction (p<0.58) there was a statistically significant difference of 17% between the groups, 69% for the experimental group compared to 52% for the control group. There was also a statistically significant difference in permanent reaction (p< .01), 17% for the control group and 44% for the experimental group. The group that used the state-based programming system showed a higher completion ratio.

5.2 Artifact Analysis of Open Assignment Programming

The free programming assignment was analyzed based on the Cyclomatic Complexity *M* value to assess the complexity of the programs created by the students. **Table 5** shows the results of comparing the Cyclomatic Complexity *M* between the two groups.

Table 5. Difference in Cyclomatic Complexities *M*

Group	Median *M*	Q	Wilcoxon U
Control	4	2.5-4.5	45.5*
Experimental	6	5-6	

Medians and interquartile ranges (Q) are based on intervals containing the 25, 50 and 75 percentiles, respectively
Significant level: * p<.05

The medians were found at 4 for the control group and 6 for the experimental group, showing a statistically significant difference (Wilcoxon rank-sum test, p-value <0.05). **Fig. 9** shows the two groups' scripts that are the most complicated programs. The control group's program had 7 layers of nesting, and tiles appeared that made it difficult for the students to view the SXGA computer screen they used. Errors such as 'unreachable code' ("exitRepeat" command marked as *1 in **Fig. 9**) and the 'if sentence without if' (marked as *2 in **Fig. 9**) appeared in the programs. On the other hand, it was easy for students of the experimental group to view the computer screen because the program allows for arranging the state script anywhere they wanted, and errors did not appear in the program.

Control group (*M*=10)

```
MYUCARC >> script1
   [   (self getSwitch: 2) ifTrue: [
         self turnRight: 1.
         (self getSwitch: 3) ifTrue: [
             [  self forward: 1.
               (self getSwitch: 2) ifTrue: [
                  self turnRight: 10.
                  [  self forward: 1.
                     (self getSwitch: 3)  ifTrue: [
                        [  self wait: 1.
                           (self getSwitch: 2)  ifTrue: [
                              self turnLeft: 10.
                              [   self forward: 1.
                                 (self getSwitch: 3) ifTrue:[
                                    [  self wait: 1 ] repeat ]
                              ] repeat.
                              self exitRepeat]. "*1"
                        ] repeat.
                        self exitRepeat]. "*1"
                  ] repeat.
                  self exitRepeat]. "*1"
               true ifTrue: []."*2"
            ] repeat.
            self exitRepeat]. "*1"
      (self getSwitch: 3) ifTrue: [
         [  self turnLeft: 1.
            (self getSwitch: 2) ifTrue: [
               self wait: 15]
         ] repeat.
         self exitRepeat]. "*1"
   ] repeat
```

Experimental group (*M*=9)

```
MYUCAR >> state1
   self forward: 2.
   self back: 2.
    (self getSwitch: 2) ifTrue: [
       self transitTo: #state2]
MYUCAR >> state2
   self turnRight: 1.
    (self getSwitch: 3) ifTrue: [
       self transitTo: #state3]
MYUCAR >> state3
   self turnLeft: 1.
    (self getSwitch: 2) ifTrue: [
       self transitTo: #state4]
MYUCAR >> state4
   self wait: 5.
   self forward: 1.
   self turnRight: 3.
    (self getSwitch: 3) ifTrue: [
       self transitTo: #state5]
MYUCAR >> state5
   self forward: 2.
   self turnRight: 1.
    (self getSwitch: 2) ifTrue: [
       self transitTo: #state6]
MYUCAR >> state6
   self wait: 5.
   self back: 1.
   self turnLeft: 3.
    (self getSwitch: 3) ifTrue: [
       self transitTo: #state7]
MYUCAR >> state7
   self wait: 6.
   self turnRight: 10.
    (self getSwitch: 2) ifTrue: [
       self transitTo: #state8]
MYUCAR >> state8
   self wait: 6.
   self turnLeft: 1.
    (self getSwitch: 3) ifTrue: [
       self transitTo: #emptyState]
```

Fig. 9. Specially complicated programs written by students.

5.3 Program Comprehensibility

Table 6 shows the results of comparing the program reading ability between the two groups. A total of 10 questions were measured on a 10-point scale.

Table 6. Mean scores for each question of reading test

Concept	Group	*M* (SD)	t-value
Total	Control	7.13 (2.03)	2.710*
	Experimental	8.70 (0.78)	
Repetition	Control	9.75 (0.91)	0.001
	Experimental	9.75 (0.90)	
Temporary reaction	Control	7.35 (1.39)	1.420
	Experimental	8.01 (1.03)	
Permanent reaction	Control	6.90 (2.09)	2.934**
	Experimental	9.39 (0.75)	

Significant level: * p<.05, ** p<.01

According to the results of analyzing program reading ability, the mean of the experimental group was 8.70, significantly higher than the mean of 7.13 for the control group. The two groups did not show a significant difference in reading ability for repetition and temporary reaction but showed a significant difference in permanent reaction, 9.39 for the experimental group and 6.90 for the controlled group. Since the experimental group learned a state-based

programming system suitable for middle school students, it showed a higher understanding for overall aspects of programming than the control group that learned the general programming system. Educational programming language is used in teaching programming to middle school students because it is hard for students to understand general programming. However, the use of the tool developed in this study for teaching programming to students sparked students' interest in programming and enhanced their programming performance. The development of the tool is significant in that it helps students be exposed to programming more easily at the beginning stage.

Table 7 shows the results of the Wilcoxon rank sum test for repetition and temporary reaction.

Table 7. Differences of average scores for each question group with reaction types

Reaction type	Group	Median	Q	Wilcoxon $U_{16\text{-}16}$
All	Control	7.8	7.2-8.3	76.2*
	Experimental	9.1	8-9.4	
No and temporarily	Control	3.0	2.7-3.3	112.5
	Experimental	3.0	3-3.3	
Permanent	Control	5.0	4.3-5.7	75*
	Experimental	5.7	5.2-6	

Medians and interquartile ranges (Q) are based on intervals containing the 25, 50 and 75 percentiles, respectively
Significant level: * p<.05

According to the results, the two groups showed a significant difference only in permanent reaction on which this study focused ($p < 0.05$, Fisher's exact test). It was confirmed that the state-based programming was more effective in permanent reaction that increases the complexity of a program. Students could solve a tougher program more easily because they understood the program well. Those results support that the state-based programming can contributes to improving students' problem solving abilities.

6. Discussion and Conclusion

This study attempted to seek a better way to use robot programming for educational activities because students have such a keen interest in the subject. It is easy to choose robot programming in education because it motivates students not only to learn programming but also doing so helps them develop their critical and creative thinking abilities. However, students who are beginners must from the start understand the concept of reactivity very well to write programs suitable for the real world. This study considered reactivity as an integral part of procedural programming, and developed State–Based Squeak to support robot programming. In addition, this paper examined the effect of state-based programming system through an experiment.

According to the results, the use of the state-based programming system was found more effective when students learned or implemented difficult concepts. The effect was especially higher on complex contents which require the use of many behaviors to implement a program.. In addition, to achieve effectiveness, teachers explained as follows:

"State-based programming helps students divide the robot movement into small units."

This explanation confirms that teachers were also aware of the second advantage of the state-based programming system, "an appropriate standard is necessary for breaking up the program into units."

"It was hard to explain the concept of 'states' and it is necessary to arrange the relation of the states in a 2-dimensional diagram before students made their programs. It was also hard to evaluate the project carried out by students who were asked to make their programming system complicated."

To examine the effect of the state-based program, students were asked to write a program that would have the robot execute as many behaviors as possible. When teachers evaluated the projects completed by the students, they sometimes had difficulty evaluating the quality of a robot's movement if the purpose of the robot movement was not clear. This difficulty was indicated by the teachers. Therefore, we suggest to 'explain the concept of states more easily' considering the 'advantage of breaking up the program into small units' and to 'develop a suitable task for state learning'.

This study examined the effect of state-based programming, but the number of the participants was small and the capabilities of Rtoys were limited. For an education based on state-based programming, it is required to keep the following in mind:

First, state-based programming is a programming suitable for reactivity measurement and control. For straight-line programs, programmers using state-based programming must divide the system into more units than is needed for procedural programming. This higher complexity is a disadvantage of state-based programming.

Second, state-based programming has the disadvantage of decreasing the readability of program code as the task to be done becomes more complicated. State-based programming has the advantages of eliminating the limitations of procedural programs and can control program flow more freely by state transition. However, as with Dijkstra's "Go To Statement Considered Harmful", arbitrary jumping makes programming more difficult in reading and tracing the execution steps in it. As the conditions to be considered become more, it is possible that the number of state gets too many to handle. It is just as so called "state explosion".

The relation of states can be explained in a 2-dimensional diagram to trace complicated state transitions easily. This study solved the complexity problem by explaining each state in a 2-dimensional screen in the script through Rtoys, which is a visual programming environment.

As the relation of state explosion, a hierarchical structure will be needed. In a practical state diagram, such as a *Harel Chart* or modern *UML*, states are classified into a hierarchical structure like concurrent states or substates [42]. It is technically possible to run concurrent states in State-Based Squeak, but we did not enable it on Rtoys in order to provide the only concepts that novice programmers can understand. However, it is necessary to examine more diverse effects of state-based programming by teaching classified states in a hierarchical structure to novice programmers and asking higher-level learners such as high school students to deal with concurrent states.

The State-Based Squeak and its programming system were designed and developed, so students are expected to perform diverse logic activities during their problem solving process. The significance of this study lies in its ability to help students learn robot programming more easily and develop their scientific thinking by leading them to solve problems in the same way as computer scientists do.

References

[1] Alan Kay, "Squeak Etoys, Children & Learning," *VPRI Research Note* RN-2005-001, Viewpoints Research Institute, California, 2005.

[2] DaiYoung Kwon, IlKyu Yoon, & WonGyu Lee, "Design of programming learning process using hybrid programming environment for computing education," *KSII Transactions on internet and information systems*, vol.5, no.10, pp.1799-1813, 2011. Article (CrossRef Link).

[3] Dagdilelis, V., Satratzemi, M, and Kagani, K., "Teaching (with) robots in secondary schools: some new and not-so-new pedagogical problem," in *Proc. of the 5th IEEE Int. Conf. on Advanced Learning Technologies* , pp. 757-761, 2005. Article (CrossRef Link).

[4] Jacky Baltes & John Anderson, "Introductory programming workshop for children using robotics", *Int. Journal of Human-Friendly Welfare Robotic Systems*, vol. 6, no .2, pp. 17-26, 2005.

[5] Ministry of Education, Culture, Sports, Science and Technology, "Section 8 Technology and Home Economics," *New government curriculum guideline of junior high school*, http://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2011/04/11/1298356_9.pdf, 2008.

[6] Shinichiro Kodaira, Hiroshi Sakamoto, & Yasuo Harigaya, "Verification of learning effect based on lesson practices of 'measurement/control by computer program' learning using autonomous type robot as a teaching material" [in Japanese], *Journal of the Japan Society of Technology Education*, vol. 51, No. 4, pp. 285-292, 2009.

[7] Alimisis, D., Moro, M., Arlegui, J., Pina, A., Frangou, S., and Papanikolaou, K., "Robotics & Constructivism in Education: the TERECoP project", In Kalas, I. (ed.) in *Proc. of the 11th EuroLogo Conference*, 2007.

[8] Maria Kordaki, "A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation," *Computers & Education*, vol. 54, no. 1, pp. 69-87, 2010. Article (CrossRef Link).

[9] Geoffrey Biggs and Bruce A. MacDonald, "Specifying Robot Reactivity in Procedural Languages", in *Proc. of the 2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp.3735–3740, 2006. http://dx.doi.org/10.1109/IROS.2006.281755 Article (CrossRef Link).

[10] Fred Martin, "Ideal and real systems: A study of notions of control in undergraduates who design robots," In Kafai, Y. and Resnick, M (ed.), *Constructionism in Practice*, Erlbaum, Mahwah, NJ, 1994.

[11] Hiroyuki Aoki, Hirofumi Nishigaya, Shuji Kurebayashi, and WonGyu Lee, "The implication of robot programming classes promoting structured thinking reveals their difficulties and remedies," in *Proc. of The KACE and the KAIE Winter Conf.*, vol. 12, no. 1, pp. 105–110, 2008 [in Korean].

[12] Joseph J. Pfeiffer, "Altaira: a rule-based visual language for small mobile robots", *Journal of Visual Languages & Computing*, vol.9, no.2, pp.127-150, 1998. Article (CrossRef Link).

[13] Biggs, G. and MacDonald, B. A., "On specifying reactivity in robotics," in *Proc. of the Australian Conf. on Robotics and Automation*, 2005.

[14] Peta Wyeth, Mark Venz, & Gordon Wyeth, "Scaffolding children's robot building and programming activities," In D. Polani et al. (Eds.), *RoboCup 2003*, *LNAI 3020*, pp. 308-319, 2004. Article (CrossRef Link).Randall D. Beer, Hillel J. Chiel, & Richard F. Drushel, "Autonomous robotics to teach science and engineering," *Communications of the ACM*, vol.42, no.6, pp. 85-92, 1999. Article (CrossRef Link).

[15] Seymour Papert, *Mindstorms: Children computers and powerful ideas*, New York: Basic Books, 1980.

[16] Fadi P. Deek, Starr Roxanne Hiltz, Howard Kimmel, and Naomi Rotter, "Cognitive assessment of students' problem solving and program development skills," *Journal of Engineering Education*, Vol.88, No.3, pp. 317–326, 1999.

[17] Elizabeth Mauch, "Using technological innovation to improve the problem-solving skills of middle school students: Educators' Experiences with the LEGO Mindstorms Robotic Invention System," *Clearing House*, vol.74, no.4, pp. 211–214, 2001. Article (CrossRef Link).

[18] Fred Martin, "Real robots don't drive straight," in *American Assoc. for the Advancement of Artificial Intelligence Spring Symposium on Robots and Robot Venues*, 2006.
[19] Jaime Montemayor, Allison Druin, Allison Farber, Sante Simms, Wayne Churaman, & Allison D'Amour, "Physical programming: designing tools for children to create physical interactive environments," *CHI 2002, ACM Conf. on Human Factors in Computing Systems, CHI Letters*, Vol.4, No.1, pp. 299-306. Article (CrossRef Link).
[20] Biggs, G. and MacDonald, B. A., "A survey of robot programming systems," in *Proc. of the Australian Conference on Robotics and Automation*, 2003.
[21] Merredith Portsmore, "ROBOLAB: Intuitive robotic programming software to support lifelong learning," *APPLE Learning Technology Review*, pp. 26–39. 1999.
[22] James Floyd Kelly, *LEGO MINDSTORMS NXT-G Programming Guide*, Apress, New York, NY, 2007.
[23] Wiedenbeck, S., Ramalingam, V., Sarasamma, S., and Corritore, C. L., "A comparison of the comprehension of object-oriented and procedural programs by novice programmers," *Interacting with Computers*, Vol. 11, Issue 3, pp. 255-282, 1999. Article (CrossRef Link).
[24] Roy D. Pea, "Language-independent conceptiual 'bugs' in novice programming," *Journal of Educational Computing Research*, Vol.2, No.1, pp.25-36, 1986. Article (CrossRef Link).
[25] Woodfield, S. N, Dunsmore, H. E., and Shen V. Y., "The effect of modularization and comments on program comprehension," in *Proc. of the 5th Int. Conf. on Software Engineering,* pp.215-223, 1981.
[26] Resnick, M., and Ocko, S., "LEGO/Logo: Learning Through and About Design," In I. Harel and S. Papert (eds.), *Constructionism*, Norwood, NJ: Ablex Publishing, 1991.
[27] Begel, A., *LogoBlocks: A Graphical Programming Language for Interacting with the World*, Electrical Engineering and Computer Science Department, MIT, Boston, MA, 1996.
[28] Fred Martin, Bakhtiar Mikhak, Michel Resnick, Brian Silverman, & Robbie Berg, "To mindstorms and beyond," Allison Druin, James A. Hendler (eds.) *Robots for kids: exploring new technologies for learning*, Morgan Kaufmann, San Francisco, pp. 9-33, 2000.
[29] Christopher Michael Hancock, Real-time programming and the big ideas of computational literacy, Thesis (Ph. D.) Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 2003.
[30] Jim Gindling, Andri Ioannidou, Jennifer Loh, Olav Lokkebo, & Alexander Repenning, "LEGOsheets: a rule-based programming, simulation and manipulation environment for the LEGO programmable brick," in *Proc. of Visual Languages*, pp. 172-179, 1995. Article (CrossRef Link).
[31] Yoshinori Haga, "WonderBorg and BN-1," *Journal of Robotics and Mechatronics*, vol. 14, no. 1, pp. 69-73, 2002.
[32] Stinckwich, S., Lemaignan, S., and Saidani, S., "SqueakBot: a Pedagogical Robotic Platform," *The 5th Int. Conf. on Creating, Connecting and Collaborating through Computing*, pp. 137-144, 2007. Article (CrossRef Link).
[33] Gonzalo Zabala, Ricardo Morán, and Sebastián Blanco, "Arduino Etoys: A programming platform for Arduino on Physical Etoys," in *Proc. of the 1st Int. Conf. on Robotics in Education, RiE2010*, pp. 113-117, 2010.
[34] Young-Jin Lee, "Empowering teachers to create educational software: A constructivist approach utilizing Etoys, pair programming and cognitive apprenticeship," *Computers & Education*, vol. 56, pp. 527-538, 2011. Article (CrossRef Link).
[35] Clifford J. Peshek and Jeffery Graham, "Recent developments and future trends in PLC programming languages and programming tools for real-time control," *Cement Industry Technical Conf., 1993. Record of Conf. Papers., 35th IEEE*, pp.102-113, 1993. Article (CrossRef Link).
[36] Edgar H. Bristol, "Redesigned state logic for an easier to use control language", *World Batch Forum in Toronto*, pp. 1-12, 1995.
[37] Robert W. Lewis, *Programming Industrial Systems Using IEC 1131-3: Revised Edition*. IEE Books, London, 1998.
[38] Mark Guzdial and Kim Rose, *Squeak: Open Personal Computing and Multimedia*, Prentice-Hall,

New Jersey, 2001.
[39] Fred Martin, *Robotics Explorations: A Hand-on Introduction to Engineering*, Prentice-Hall, New Jersey, 2001.
[40] McCabe, T. J., "A complexity measure," *IEEE Trans. on Software Engineering*, vol. se-2, no. 4, pp. 308–320, 1976. Article (CrossRef Link).
[41] David Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, Vol. 8, Issue 3, pp. 231–274, 1987. Article (CrossRef Link).

Hiroyuki Aoki is a Ph.D. candidate at the Department of Computer Science Education, Korea University, South Korea. He received his M.A. degree in Education from University of Tsukuba, Japan and Tokyo Gakugei University, Japan in 1999 and 2005 respectively. His research interests are programming education and development of digital teaching learning material.

JaMee Kim is a research professor at Korea University, Department of Computer Science Education in Seoul, Korea. She received the Ph. D degree from Korea University, in the department of Computer Science Education, in 2011. She received the B.A and M.A degrees in Education Evaluation from Ewha Womans University, Seoul, in 1992 and 1995. Her recent research interests are computer science education, education evaluation, education informatization evaluation, and robot programming research.

Yukio Idosaka is a teacher of "technology and home economics" at Iitakahigashi Junior High School, Japan. He is currently working towards the Ph.D degree at Osaka Electro-Communication University. He received his B.A degree in Technology Education form Mie University. His research interests include computer science education and autonomous mobile robots in technology education.

Toshiyuki Kamada is an Associate Professor at Aichi University Education, Japan. His M.S degree in Computer Science from Keio University. Currently, he is a member of the editorial board of Journal of Information Processing Systems, Korea Information Processing Society. His research interests include computational thinking in technology education, educational data mining, and distributed computing systems.

Susumu Kanemune is a professor in Osaka Electro-Communication University, Japan. He received his M.S degree and PhD degree from University of Tsukuba in 1989 and 2004 respectively. His research interest is programming language and computer science education. He is a member of the ACM, IEEE, and Information Processing Society of Japan.

WonGyu Lee is a professor at Korea University, Department of Computer Science Education in Seoul, Korea. He received the M.S and Ph.D. degrees in Computer Science from University of Tsukuba. From 1993 to 1996 he was a senior researcher at Korea Arts and Culture Service. He was the president at Korea Association of Computer Education in 2002. He was the chief director of Creative Informatics & Computing Institute at Korea University from 2007 to 2011. He made many progresses in the research of computer science education. Currently, he is a dean of students of Korea University. His research interests are computer science education, especially information retrieval, database, and educational programming.