Empirical Studies of Agile Software Development to Learn Programming Skills

Yasuyo KOFUNE
Osaka Prefectural Yodogawa Technology High School
Osaka, Japan

and

Takahiro KOITA
Doshisha University
Kyoto, Japan

ABSTRACT

This paper presents a programming education support method based on Agile Development that encourages and builds on communication between students. Through mutual discussion, students using our approach transform their ideas into software and cooperate to write a program. The students complete the software through repetition and programming. Before completing the software program, the students learn to solve problems by working together. The students are encouraged to think and share ideas, and gain experience writing software. With this approach, students not only learn how to write programs, but also increase their logical thinking, problem-solving, and communication skills.

Keywords: Programming Education, Agile Development, Mind Map, UML, Pair Programming

1. RESEARCH GOAL

Our research goal is to inspire students to voluntarily study programming and enjoy creating software and, in this way, gain the knowledge and skills needed to successfully work as engineers after graduation. Towards this goal, through a process of students cooperatively creating software, our approach using Agile Development provides a learning support method that can improve students' logical thinking, programming, problem-solving, and communication abilities.

2. BACKGROUND AND MOTIVATION

High school students in Japan are required to take an Information Study class, which provides instruction in practical skills, computer science, and the social aspects of computing. In Information Study class, students learn not only how to use the computer, but also algorithms, programming, and problem solving [1].

Most Japanese high school students, however, cannot write a program on their own. Japanese students receive considerable instruction on the grammar of programming language. Using such software as Scratch, they learn algorithms and problem solving [2], but most class time is spent studying programming language grammar. In programming class, the teacher explains the meaning of each order using a sample program of the text and the students learn how to write each order. Each student works individually and rarely has the opportunity to write a program through trial and error. Taught in this way, many students cannot understand the sample text programs and find it difficult to learn programming [3][4]. Hence, they find programming uninteresting and are not inspired to learn.

The existing techniques for teaching programming languages in Japan are not sufficient to support learning due to the lack of hands-on experience. Moreover, many students have difficulty solving problems on their own or conveying their ideas to others. However, logical thinking and problem solving are crucial to learning programming [4][5]. Understandably, because these students find learning programming an unpleasant experience and one in which they cannot express their own ideas, they do not feel a sense of accomplishment and, therefore, have little interest in voluntarily studying programming.

Active employment as an engineer requires logical thinking and problem-solving abilities in addition to programming skills. To acquire these abilities, students should have a desire to learn programming languages. Even with programming instruction, it is difficult for a student to be able to actually write a program. Improving programming skills requires trying it by oneself. However, this would require much more learning time than is available during school hours. When students wish to learn and improve their programming skills, and are able to experiment through trial and error, they gain considerable and valuable experience.

3. OUR APPROACH

Agile Development can be a useful tool for increasing the logical thinking and problem-solving abilities of students. With our approach, a student who is weak in programming voluntarily participates and learns in a group with other students. The students share work and, in this cooperative learning environment, necessarily exchange opinions. Each student conveys their ideas to the other students and they share each other's thoughts. This unifying of ideas is crucial to the development of the software the students are trying to create. This cooperative development of software by the students demonstrates the effectiveness of Agile Development [6].

Agile Development is an umbrella term for several iterative and incremental software development methodologies. The following two points illustrate the Agile Manifesto's main principles [7]:

・The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

・Working software is the primary measure of progress.

Under the Agile Manifesto's principles, students can study program languages, software design, and user interfaces.

First, students inspect the structure of a certain completed software and then discuss how to divide the software into small functional units to examine the function of their software.
To complete their software, the students must share a mutual idea and all members need to understand the software in its entirety. Therefore, the students create a *Mind Map* of their software's structure and function. Mind Map is Tony Buzan's method of diagramming to visually outline information [8].

Next, the students learn to model software using the Unified Modeling Language (UML). UML is a standard general-purpose modeling language for object-oriented programming [9].

The students first create a UML use case diagram based on the Mind Map showing the software's structure. The students determine whether each function is important or not. Moreover, the students design the software structure using the UML object and class diagrams. The students describe the exchanges between classes using a UML sequence diagram. With UML, it is necessary that all students cooperate and write the program. To write UML, the students need to understand the required specifications and be able to think logically. Learning UML is extremely beneficial [10][11].

The students write programs based on their UML diagram. Each student's program is shared among the students. Therefore, each student explains the program that they created to the other students. Using pair programming, the students can cooperate and write a program [12][13]. Pair programming is a technique for two-person program development and is often used by Agile Development. Finally, the students join the programs with the functional units and complete their software. Figure 1 shows the overall flow of the students' software development under our approach.

The Agile Development technique values a member's communication and repeats the development process. Similarly, students of our approach also reexamine the entire software, test each function of the program, and develop mutual work conditions.

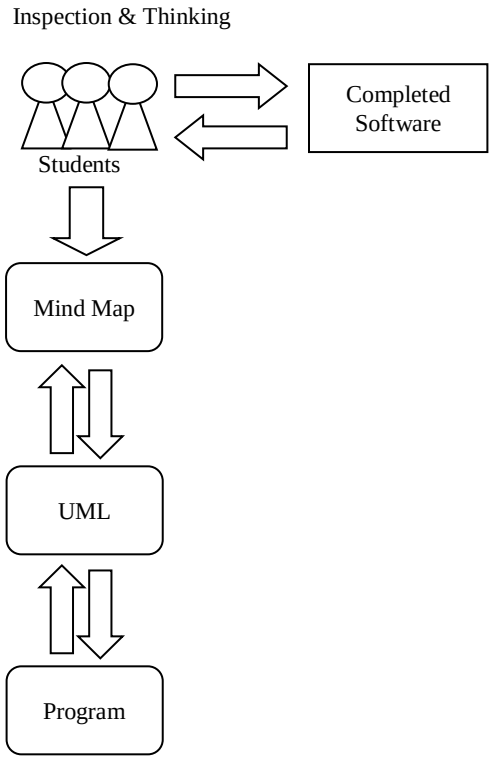

Figure 1: Overall flow of our approach

4. EXAMPLE OF OUR APPROACH

Our approach divides a development process into four phases: Requirements specification; Mind Map creation, UML, and programming. However, it is not necessary to follow any particular order and students can freely return to former stages and do them over again. Principles of Agile Development include information sharing, short-term development, and testing [7]. Our approach also considers information sharing and short-term development important, particularly because short-term development increases the students' desire to learn. The following is an example of our approach. In this example, students develop a control program for a line tracing car.
A line tracing car is a model car that runs automatically along a black line drawn on white paper, as illustrated in Figure 2.

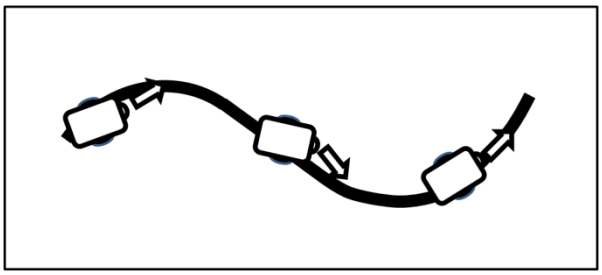

Figure 2: Line tracing car

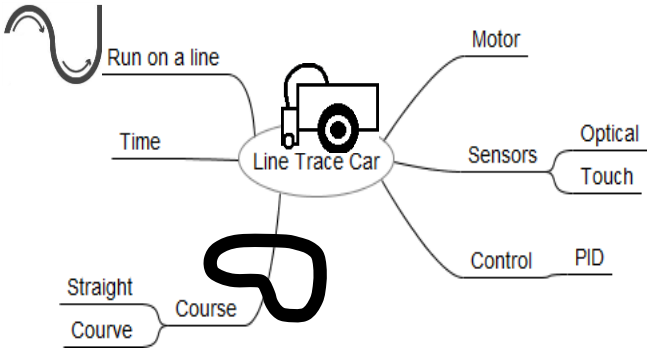

Figure 3: Example of Mind Map

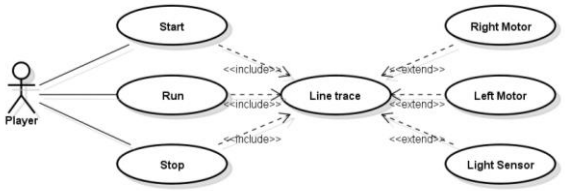

Figure 4: Example of UML use case diagram

The first phase of making the software is determination of the required specifications. The students discuss the functions necessary for the line tracing car and share their ideas. The students write the result of this cooperative discussion in a Mind Map and later write the program for each function. To that end, it is necessary to understand the software in its entirety, which underscores the importance of collaborating and sharing thoughts. Figure 3 shows an example of the Mind Map for this project.

The structure of the software using UML is decided in the next step. During this process, the students may reexamine the requirement specifications. Using a UML use case diagram, the students confirm what is necessary for the function of the line tracing car. Figure 4 shows the UML use case diagram for this example. Using the UML object and class diagrams, the students are able to consider the whole software structure. Figures 5 and 6 show the UML object diagram and UML class diagram, respectively, for this example. Using a UML sequence diagram, the students can confirm the movement of the software. Figure 7 shows the UML sequence diagram. The students cooperate through pair programming and write the program. In this way, the students understand not only the portion of the program that they wrote individually, but also the parts written by the other students.

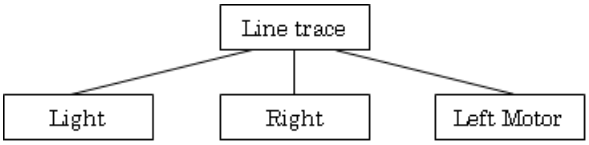

Figure 5: Example of UML object diagram

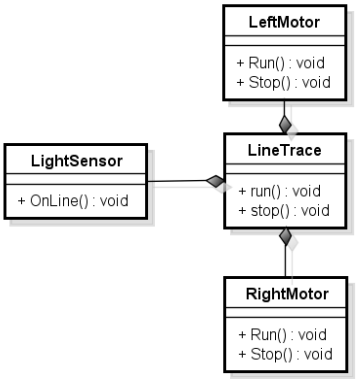

Figure 6: Example of UML class diagram

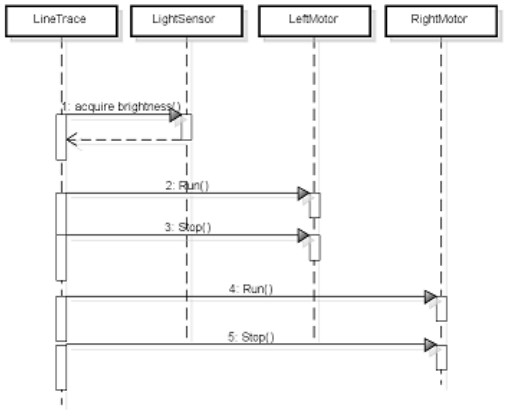

Figure 7: Example of UML sequence diagram

The students combine the parts of the program that each student wrote to complete the software to move a line tracing car. The students then test whether the line tracing car responds according to their demands. The students repeat the requirements confirmation, programming, and test runs, and then finalize the software.

5. DISCUSSION

We used the line tracing car example to illustrate our approach with Agile Development. However, in practice, the teacher does not determine the problem the students will address; the students present their own ideas and create the software that they want. Students want to create software that they already know, for example, software that they use daily or software that

controls robot motion. To inspire voluntary study and participation, it is important to allow students to do things that they want to do at their own pace. In this way, they will continue learning the difficult skills required for software design and programming.

The learning environment in a class that adopts our approach is not based on the teacher's lesson plan, but rather on the ideas of the students. Furthermore, the students must meet a great variety of demands, which keeps the subject matter interesting. Such a class requires a high level of skill on the part of the teacher, but the relationship of the teacher to the students is the same as the relationship between the students; that is, they teach and learn from each other.

The students confront many problems and improve their programming skills and problem-solving and communication abilities through trial and error. However, this may be difficult for students who are not yet adept at reading, writing, and calculation. Therefore, we should devise a support method to assist teachers working with less experienced students.

6. CONCLUSIONS

With our approach, students confront many problems while creating software, which may or may not be solved. Through trial and error, failure and success, the students gain invaluable hands-on experience. This provides students with the skills necessary to write software programs and strengthens their logical thinking, problem-solving, and communication abilities.

The students also stay motivated to learn because they are creating the kind of software that they want. Inspired by the opportunity to make their ideas reality, students will voluntarily participate and increase their computer programming skills.

REFERENCES

[1] Japanese government guidelines for high school education (Information subject area), Ministry of Education, Culture, Sports, Science and Technology-Japan (MEXT), 2010.

[2] M. Resnick, "Scratch Programming for All," ACM Communications, Vol. 52, Issue 11, pp. 60-67, 2009.

[3] E. Lahtinen, K. Ala-Mutka and H. Järvinen, "A Study of the Difficulties of Novice Programmers," Proc. of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE, Vol. 37, Issue 3, pp. 14-18, 2005.

[4] K. Ala-Mutka, "Problems in Learning and Teaching Programming - A Literature Study for Developing Visualizations in the Codewitz-Minerva Project, " 2003.

[5] A. Gomes and A.J. Mendes, "Problem Solving in Programming," Proc. of the PPIG, 2007.

[6] V. Devedžić and S.R. Milenković, "Teaching Agile Software Development: A Case Study," IEEE Transactions on Education, Vol. 54, Issue 2, pp. 273-278, 2011.

[7] Manifesto for Agile Software Development, http://www.agilemanifesto.org/.

[8] Tony Buzan - Inventor of Mind Mapping, http://www.tonybuzan.com/about/mind-mapping/.

[9] UML: Object Management Group, http://www.uml.org/.

[10] I. Diethelm and L. Geiger, A. Zündorf. "Teaching Modeling with Objects First," Proc. of the World Conference on Computers in Education, 2005.

[11] C. Starrett, "Teaching UML Modeling Before Programming at the High School Level," Proc. of the IEEE International Conference on Advanced Learning Technologies, pp. 713-714, 2007.

[12] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," Extreme Programming Examined, 2000.

[13] L. Williams and R.L. Upchurch," In Support of Student Pair-programming," Proc. of the SIGCSE Technical Symposium on Computer Science Education, pp. 327-331, 2001.