International Multidisciplinary Research Journal
ISSN: 2424-7073
Available Online at https://imrjournal.info/
Volume: III; December 2024; Page No.31-39
DOI: https://doi.org/10.47722/imrj.2001.34

Research Article

APPLYING SECOND LANGUAGE ACQUISITION (SLA) STRATEGIES TO PROGRAMMING EDUCATION: A COMPARATIVE ANALYSIS OF LEARNING APPROACHES IN EARLY JAPANESE CURRICULUM

Katsuyuki Umezawa[1]; Makoto Nakazawa[2]; Michiko Nakano[3]; and Shigeichi Hirasawa[4]

[1]Professor, Department of Informatics, Shonan Institute of Technology, Fujisawa, Kanagawa, Japan
[2]Professor, Department of Industrial Information Science, Junior College of Aizu, Aizuwakamatsu, Fukushima, Japan
[3]Honorary Professor, Faculty of Education and Integrated Arts and Sciences, Waseda University, Shinjuku, Tokyo, Japan
[4] Honorary Professor, Research Institute for Science and Engineering, Waseda University, Shinjuku, Tokyo, Japan

ARTICLE INFO

Article History:

Received 15.08.2024
Accepted 15.10.2024
Published 15.12.2024

Keywords:

programming language, second language, SLA, blended learning, CFER

ABSTRACT

In recent years, the "government curriculum guidelines" have been revised in Japan to emphasize the early introduction of English and programming education. Foreign language activities were included in elementary school curriculum as for English education, and full-fledged information education began in elementary and secondary education as for programming education. There are two specific methods, i.e., "learning concepts themselves" and "learning from examples," for learning concepts in programming learning. However, no unified view exists as to which of these two methods should be started in the introductory stage of learning. Many research have been conducted on programming learning strategies. The authors thought that programming languages and English can be regarded as common terms of "second language," which is a language other than their mother tongue. Moreover, the authors thought that the second language acquisition (SLA) approach, which has been extensively studied in programming language learning, could be applied. In this paper, the authors classify the subjects they are in charge of using the level classification method of programming language learning defined in previous research. This categorization indicates that the author's course covers all comprehension levels defined in the second language acquisition project in blended learning (SLA-aBLe) project (from minimal comprehension to advanced fluency).

1. Introduction

In 2016, the "government curriculum guidelines" in Japan emphasized the early introduction of English and programming education toward globalization and the internationalization of

education. Regarding English education, elementary school foreign language activities were included in the curriculum, and foreign language activities started from the third grade of elementary school. In the 5th and 6th grades, English education was treated as a new subject, namely, “foreign languages.” The quality of higher education must be evaluated internationally, and there is an urgent need to improve English language proficiency at an international level. However, programming education was introduced in elementary and junior high schools from the 2020 academic year. In the upper secondary school curriculum, the “information” subject, which includes programming, has been made compulsory from the 2022 academic year. Thus, full-fledged information education has begun in elementary and secondary education. In addition, the Ministry of Internal Affairs and Communications also promotes programming education with a policy to train 1 million people.

Previous study (*Okamoto* et al., 2013) reported that there are two concrete methods for learning concepts in programming, i.e., “learning the concepts themselves” and “learning from examples,” however, there are differing opinions on which of these two methods should be started in the introductory stage of learning. In the field of mathematics education, it has been pointed out that learning strategies that begin with concrete examples are difficult due to the difficulty in extracting abstract elements from concrete examples. However, in programming learning, it has been pointed out that learning from examples and giving students an image of what it is like to execute a program are effective.

As described above, many studies have been conducted on programming learning strategies. The authors thought that programming languages and English can be considered as “second languages,” which are languages other than one's native language. They also believed that the much-studied “second language” acquisition (SLA) approach could be applied during learning of programming languages.

The authors compared the biometric information during learning of programming languages and English. The results showed that significant differences were observed in the biometric information when solving problems in the same category, such as reading comprehension problems in English and programming languages or English composition and programming problems. In contrast, some biometric information showing significant differences were observed while solving problems of different categories. This suggested that English learning strategies could be applied to programming learning for similar problems.

In this study, the authors first describe a programming skill self-evaluation matrix (*Poss, R.* 2018) that applies programming language learning to the learning level framework of second language learning methods. Next, the authors introduce a second language acquisition project in blended learning (SLA-aBLe) (*Sun, L.*, et al., 2017), which aims to examine the effectiveness of the SLA approach for programming language learning. The authors then classify the subjects that they were in charge of in the actual classes using the level classification method of programming language learning defined in those previous studies. This classification indicates that the author's course covers all comprehension levels (from minimal comprehension to advanced fluency) defined in the SLA-aBLe project.

2. Previous work

2.1 Common European Framework of Reference for Languages (CEFR)

The Common European Framework of Reference for Languages (CEFR) (*Council of Europe,* 2020a) provides a common basis for the refinement of language syllabuses, curriculum guidelines, exams, and textbooks, among others, across Europe. The CEFR comprehensively explains what language learners must learn to use language for communication and what knowledge and skills they must acquire to be able to act effectively. The CEFR has detailed definitions for each level based on six levels, A1, A2, B1, B2, C1, and C2. The CEFR Companion Volume (*Council of Europe,* 2020b) also presents key CEFR messages in a user-friendly format and includes all CEFR explanatory descriptors. Teachers and teacher educators will find it easy to access the CEFR Companion Volume as an updated framework for educational use of the CEFR for learning, teaching, and assessment. The Companion Volume references chapters from the 2001 edition as necessary. Table 1 presents an overview of the CEFR levels. Although the table 1 only presents the classification of levels, the CEFR actually provides specific descriptions for each level. For example, the A1 level of the category "listening" of "UNDERSTANDING" is described as follows: "I can recognize familiar words and very basic phrases concerning myself, my family, and immediate concrete surroundings when people speak slowly and clearly."

Table 1 - CEFR Leveling

Group	Group name	Level	Level Name
A	Basic User	A1	Breakthrough
		A2	Waystage
B	Independent User	B1	Threshold
		B2	Vantage
C	Proficient User	C1	Effective operational efficiency
		C2	Mastery

2.2 Programming skills self-assessment matrix

The programming skills self-assessment matrix (*Poss, R.* 2018) is inspired by the CEFR table to assess natural language proficiency and is applied to programming language acquisition (see Table 2).

Table 2 - Classification of programming skill self-assessment matrix

Classification	Minor classification	A1	A2	B1	B2	C1	C2
Writing	Writing code	✓	✓	✓	✓	✓	✓
	Refactoring						
	Embedding in a larger system						
Understanding	Reusing code						
	Explaining / discussing code						
Interacting	Exploring, self-learning						
	Mastery of the environment						
	Troubleshooting						

It characterizes a programmer's proficiency in a particular programming language (columns) in the context of various programming activities (rows). Similar to the CEFR, learners are roughly divided into three levels: basic user (A), independent user (B), and proficient user (C). Each of these three levels is further divided into two levels (A1, A2, B1, B2, C1, C2), corresponding to testable milestones in programming language acquisition.

This table can be used in a number of ways. For example, it is possible to evaluate one's own level (set a different skill level for each activity) for each row (for each activity). It is also possible to determine the lowest level of a programming language by looking at the requirements column by column from left to right. Moreover, it is possible to determine the most developed skill for each column from right to left. It is also possible to assess one's relative proficiency in different programming languages.

Table 3 presents the details of the "writing code" row (check marked cell) in Table 2 (note that the vertical and horizontal sides are reversed). In addition, although the text about the "writing code" is quoted here, the document (*Poss, R.* 2018) describes all the cells presented in Table 2 in detail.

Table 3 - Level division for writing code

Level	Description
A1	I can produce a correct implementation for a simple function, given a well-defined specification of desired behavior and interface, without help from others.
A2	I can determine a suitable interface and produce a correct implementation, given a loose specification for a simple function, without help from others. I can break down a complex function specification in smaller functions.
B1	I can estimate the space and time costs of my code during execution. I can empirically compare different implementations of the same function specification using well defined metrics, including execution time and memory footprint. I express invariants in my code using preconditions, assertions and post-conditions. I use stubs to gain flexibility on implementation order.
B2	I use typing and interfaces deliberately and productively to structure and plan ahead my coding activity. I can design and implement entire programs myself given well-defined specifications on external input and output. I systematically attempt to generalize functions to increase their reusability.
C1	I can systematically recognize inconsistent or conflicting requirements in specifications. I can break down a complex program architecture in smaller components that can be implemented separately, including by other people. I can us existing (E)DSLs or metaprogramming patterns to increase my productivity.
C2	I can reliably recognize when under specification is intentional or not. I can exploit under-specification to increase my productivity in non-trivial ways. I can devise new (E)DSLs or create new metaprogramming patterns to increase my productivity and that of other programmers.

2.3 SLA-aBLe project

The aforementioned programming skill self-assessment matrix is also based on the CEFR for SLA and applied to programming language acquisition. Many studies apply SLA methods to programming language acquisition. The SLA-aBLe project aimed to examine the effectiveness of the SLA approach for programming language learning. In this project, researchers focused on the

commonalities between programming languages and natural languages and applied successful examples of SLA to programming language education from the perspective of teaching methods. The results were higher when the educational method for SLA was applied. As the evaluation items at this time, the intrinsic motivation index (interest, tension, etc.) and the self-evaluation type workload index (effort, frustration, etc.) are evaluated through questionnaire.

Table 4 presents a comparison of the currently (in 2015) applied learning method and the SLA-aBLe method shown in reference (*Sun, L.*, et al., 2017). From the table, detailed points of improvement can be observed, including basic parts such as simple explanation using multiple images, group work to make students learn more proactively, and providing presentation opportunities. In addition, Table 5 presents specific examples of problems used in the class exercise for the SLA-aBLe method shown in reference (*Frederick, C. M.*, et al., 2017). From this table, it can be seen that as the level of the learner increases, the content becomes more suitable for practice.

Table 4 - Comparison of the current (in 2015) blended learning and the SLA-aBLe method

	Preproduction (minimal comprehension)	**Early production (limited comprehension)**	**Speech emergence (increased comprehension)**	**Intermediate fluency (very good comprehension)**	**Advanced fluency**
Current (in 2015) Blended Learning	Few pictures and visuals. Some topics are not well explained. Not enough self testing questions in the screencasts.	There are multiple choice questions but no simple programs. Facebook is used but there is no group discussion.	Students begin reading and writing in their programming language by solving different engineering problems.	Give students more challenging problems to synthetize what they have learned.	Open-ended engineering project to challenge their understanding and expand their knowledge.
Teaching Strategies in SLA-aBLe	Use pictures and visuals; speak slowly and use simple and shorter words to draw connection between SLA and programming languages; Reinforce learning by giving more self testing questions without adding in pressure.	Reinforce learning by asking students to produce simple programs in addition to the multiple choice questions; use discussion board to encourage group discussion.	Emphasize tiered questions and ask students to do a "think, pair, share" to process the new concepts.	Emphasize compare and contrast different concepts. Allow students to explain their problem solving process.	Project presentation opportunity will be offered to students to enhance their understanding.

Table 5 - Examples of exercises

	Preproduction (minimal comprehension)	**Early production (limited comprehension)**	**Speech emergence (increased comprehension)**	**Intermediate fluency (very good comprehension)**	**Advanced fluency**
Specific SLA-based in-class exercises	Show me ⋯, Circle the ⋯, Where is the ⋯	Yes/No questions, Either/Or questions, Use 1-2 word answers, Use lists and labels	Ask why and how questions, Ask students to explain using phrase or short sentence answers	Use "What would happen if ⋯" questions, Use "Why do you think" questions	Use "decide if" exercises, Have student "retell" in his/her own words

3. Analysis of author's subjects

In this section, the authors analyze the main programming-related subjects during the COVID-19 pandemic (2020-2022) by referring to previous research. Specifically, first, an outline of the author's subject is shown. After that, the programming skill self-assessment matrix discussed in section 2.2 is used to judge the level of the author's course. Finally, the author's subject is applied using the level classification table of the SLA-aBLe project discussed in section 2.3.

3.1 Author's programming-related courses

During the COVID-19 pandemic (2020-2022), the author's main programming-related courses are as follows.

- Basic programming practice (1st year)
- Information ethics and security (2nd year)
- PBL programming practice A (2nd year)
- Compiler (3rd year)
- Software engineering (3rd year)

In addition, Table 7 in the appendix summarizes the purpose of each subject, the goal to be achieved, and the method of teaching. In 2020, real-time online classes were held using Google Meet. In 2021, a hybrid class was implemented, with real-time online classes for students who needed to stay at home, such as when a family member was sick with the coronavirus, and face-to-face classes were conducted in the classroom for students who had no health problems. In 2022, individuals have mostly returned to face-to-face classes.

3.2 Level judgment using the programming skill self-assessment matrix

The author's subject (Table 7) was compared with the programming skill self-assessment matrix classification table (Table 3), and "software engineering" can be said to be a subject aiming for the B1 or B2 level. However, it can be said that other subjects mostly belong in the A1 or A2 category. Some laboratories aim to master the C1 or C2 level if they progress to the graduation research of the fourth year. However, the author's subject up to the third year can be described as a "basic language user" according to the CEFR level classification.

3.3 Mapping the author's subjects to SLA-aBLe

The SLA-aBLe project shown in section 2.3 was said to have been implemented in a class for an introductory computing course for engineers. It can be said that the target students' programming level is almost the same as the students of the author's course shown in the previous section. Therefore, based on the educational strategies in Table 4 presented in the SLA-aBLe project, the course content of the author's course is mapped in Table 6. The author underlined the supporting statements that were considered relevant to the course content of the author's course. As presented in Table 6, all the comprehension levels (minimal comprehension to advanced fluency) of the SLA-aBLe strategy could be covered by assigning the author's subjects from the first year to the third year. It should be noted that multiple faculty members are in charge of programming-related classes in the department, so there is no need for one faculty member to cover all of the SLA-aBLe strategies. However, considering the cooperation between individual subjects, it is not meaningless for one teacher to cover all levels.

Table 6 - Applying the author's subject to SLA-aBLe

	Preproduction (minimal comprehension)	**Early production (limited comprehension)**	**Speech emergence (increased comprehension)**	**Intermediate fluency (very good comprehension)**	**Advanced fluency**
Contents of the author's course	**[Information ethics and security]** Investigate familiar incidents and accidents. Introduce security videos provided by Information-technology Promotion Agency into the class, targeting LINE and other tools that students use on a daily basis.	**[Basic prog. practice]** Learn the basics of programming using the Java language. Students who are not good at programming use the "Introduction to Java" on the paiza learning site. **[PBL prog. practice A]** This course is mainly based on group work. Experience teamwork while experiencing programming with the educational version of Minecraft.	**[Software engineering]** A team of six students will proceed with the development of a vending machine (a simulator that runs on a PC instead of a physical machine). **[PBL prog. practice A]** A group of four students (∓1 student) creates a structure by programming in the same world.	**[Software engineering]** Only after a review (explanation to teachers and teaching assistants) is carried out for each phase of development will they progress to the next phase. **[PBL prog. practice A]** Experience teamwork while experiencing programming with the Minecraft educational edition. **[Compiler]** Create a scientific calculator using recursive downward parsing among various algorithms.	**[Software engineering]** Students are finally given a presentation of the developed product and checked to see if it is made according to the specifications. **[PBL prog. practice A]** In the final week, students will hold presentations in groups to appeal the concept of the structure and the ingenuity of programming.

4. Conclusion and future work

In this study, the authors introduced the programming skill self-assessment matrix and the SLA-aBLe project as conventional studies that apply second language learning methods to programming language learning. Using the method of level classification of programming language learning defined by them, the authors classified the subjects they were in charge of during

the COVID-19 pandemic. With this classification, the authors showed that their subjects covered all comprehension levels (from minimal comprehension to advanced fluency) defined in the SLA-aBLe project.

In the future, the authors will aim to promote the mapping of the entire curriculum tree, including subjects of other teachers. In addition, since Shonan Institute of Technology has a new Faculty of Informatics from 2023, the authors will aim to proceed with the mapping of the entire curriculum tree of the new faculty. At that time, they consider whether to raise the entire department to a more advanced level of the CEFR.

5. Acknowledgments

Part of the work reported here was conducted as a part of the research project ``Research on e-learning for next-generation'' of Waseda Research Institute for Science and Engineering, Waseda University. Part of this work was supported by JSPS KAKENHI Grant Numbers JP24K06348, JP22H 01055, JP21K18535, and JP20K03082. Research leading to this paper was partially supported by the grant of ``ICT and Education'' of JASMIN.

References

Okamoto, M., Murakami, M., Yoshikawa, N., and Kita, H. (2013) Analysis of missteps in shakyo-style learning of computer programming and improvement of learning material –support and design for self-sustaining work performance and understanding through work–. Kyoto University Higher Education Research, Vol. 19, pp. 47–57.

Poss, R. (2018) How good are you at programming? a cefr-like approach to measure programming proficiency. 2014 (modified 2018).

Sun, L., Frederick, C., Ding, L,. and Rohmeyer, R. (2017) The application of second language acquisition to programming language study. American Society for Engineering Education, ASEE Annual Conference, pp. #18842 1–12.

Council of Europe. (2020a) Common european framework of reference for languages: Learning, teaching, assessment. Language Policy Unit, Strasbourg, Vol. First Edition.

Council of Europe. (2020b) Common european framework of reference for languages: Learning, teaching, assessment. Council of Europe Publishing, Strasbourg, Vol. Companion volume.

Frederick, C. M., and Sun, L,. (2017) Work in progress: Using second language acquisition techniques to teach programming - results from a two-year project. American Society for Engineering Education, ASEE Annual Conference, pp. #18825 1–13.

Appendix

Subjects taught by the author from 2020 to 2022

Table 7 presents the author's subjects in 2020-2022. In addition to the “class objectives” and “attainment goals” listed in the syllabus, the actual class methods were specifically described.

Table 7 - Author's subjects in 2020-2022

Subject	Class purpose	Goal	Class method
Basic programming practice (1st year)	The purpose of this course is to deepen the understanding of programming using the Java language. In this class, the basic structure and functions of the Java language will be learned. It also describes an important basic algorithm. Classes are conducted through a mixture of lectures and exercises.	1. Understand and learn representations, variables, and operations. 2. Understand and master inputs, branches, and iterations. 3. Understand and master arrays. 4. Understand and learn methods and classes. 5. Understand and learn basic algorithms. 6. Understand the given program. 7. Create own task programs using the grammar they have learned.	In this course, students learn the basics of programming using the Java language. Since this subject is a compulsory subject, there are students who are good and not good at programming. Based on the use of the site TechFUL, good students prepared challenge questions on TechFUL (challenge questions could be asked if they did not understand after thinking about them for 3 weeks). For students who are not good at it, it is recommended that they try "Introduction to Java" on a site called paiza learning. In this way, the author tried to provide classes that corresponded to all levels of understanding. During the COVID-19 pandemic, a question and answer corner was set up on Moodle to deal with the difficulty of asking questions in real-time online classes.
Information ethics and security (2nd year)	Students learn cryptography necessary to ensure the safety of information distribution using the Internet and techniques such as electronic signatures and authentication that apply cryptography. Moreover, they learn about the security system necessary to protect the intranet from hackers who intrude via the Internet. They also learn the basic knowledge that an information engineering engineer should acquire.	1. Know what to consider and what to pay attention to on a daily basis in the information society. 2. Learn the things to be careful about when using information and communication equipment and learn the basic measures to deal with possible dangers. 3. Acquire the ability to understand and use security-related mathematics. 4. Acquire skills related to public key cryptography and private key cryptography.	Students conduct research on familiar incidents and accidents to make security familiar to students. In addition, security videos provided by IPA are incorporated into classes to raise awareness. In addition, students are asked to check the security settings of LINE and other tools that they use on a daily basis while operating them. In addition, the latest trends in security-related academic societies are communicated to students. In the second half, they will also learn about cryptography (albeit only algorithms, not programming). During the COVID-19 pandemic, basically the same class content as face-to-face classes was conducted in a real-time online class format.
PBL programming practice A (2nd year)	This course aims to help students apply what they have learned in first-year programming classes and acquire programming skills through actual programming practice. They will also learn how to approach teamwork work through projects.	1. Improve problem-solving ability. 2. Improve programming skills. 3. Work together as a team.	This course is mainly based on group work. Students experience teamwork while experiencing programming with the Minecraft educational edition. Specifically, they will practice the Minecraft educational edition until the third week, and from the fourth week onwards, a group of four students ($\mp$1 student) will create a structure by programming in the same world. In the final week, each group will hold a presentation to appeal the concept of the structure and the ingenuity in programming. During the COVID-19 pandemic, group work was canceled, and work was done alone; however, there were students who voluntarily played multiple students simultaneously from home.
Compiler (3rd year)	The main tasks of a compiler are lexical analysis, syntactic analysis, intermediate code generation, code optimization, and object code generation. The purpose of this class is to deepen the understanding, focusing on lexical analysis and syntactic analysis. In particular, the class will focus on character string processing, which is the basis of lexical analysis.	1. Understand and master the role of the compiler and its configuration overview. 2. Understand and master character string processing. 3. Understand and master lexical analysis. 4. Understand and master parsing. 5. Implement a scientific calculator.	This subject is an elective subject. In the first half of the course, the mechanisms of compilers, such as lexical analysis and syntactic analysis, will be explained, and assignments for each item will be given to deepen understanding. The second half is programming using the Java language. At that time, students will try to create a scientific calculator using recursive downward parsing, among various algorithms. During the COVID-19 pandemic, basically the same class content as face-to-face classes was conducted in a real-time online class format.
Software engineering (3rd year)	By understanding software development as engineering, students will experience engineering development methodologies and techniques for all software development phases. Concretely, teachers will first explain the methods used in software development (class diagrams, sequence diagrams, etc.). Using these methods, the students will make a development schedule for each team and experience each phase of system development.	1. Perform system analysis. 2. Design the system. 3. Implement the system. 4. Perform testing. 5. Proceed with the development according to the development schedule that the student has planned. 6. Understand and use the methods used in software development. 7. Perform requirements analysis.	Teams of six students will work together to develop a vending machine (not a physical machine but a simulator that runs on a PC). Each team's progress is managed according to a work breakdown structure (WBS) to be decided by the students themselves at the first meeting. Each phase of development is reviewed (explanations are given to faculty members and assistants) before proceeding to the next phase. Finally, the development is presented to the students and checked to ensure that it has been produced according to the specifications. During the COVID-19 pandemic, a Google Meet meeting room was created for each group, and group work was conducted online.

How to cite this article:
Umezawa K. "et al." (2024) 'Applying Second Language Acquisition (SLA) Strategies to Programming Education: A Comparative Analysis of Learning Approaches in Early Japanese Curriculum', *International Multidisciplinary Research Journal*, Volume: III; December 2024; Page 31-39. DOI: https://doi.org/10.47722/imrj.2001.34