International Journal of Engineering Education Vol. 32, No. 3(A), pp. 1117–1127, 2016
Printed in Great Britain
0949-149X/91 $3.00+0.00
© 2016 TEMPUS Publications.

Compile Error Collection Viewer: Visualization of Compile Error Correction History for Self-assessment in Programming Education*

YOSHIAKI MATSUZAWA, MOTOKI HIRAO and SANSHIRO SAKAI
Shizuoka University Graduate School of Informatics, 3-5-1 Johoku, Hamamatsu, Shizuoka, Japan.
E-mail: matsuzawa@inf.shizuoka.ac.jp, hirao@sakailab.info, sakai@inf.shizuoka.ac.jp

We have developed CocoViewer (Compile error Collection Viewer) for learners in programming to enable them to conduct self-assessment for compiling error records. CocoViewer generates charts that show a trajectory of reducing the correction time of the compile error that is calculated by logs recorded in students' computers during a programming course. Students can see lists of charts for many kinds of compile errors, as well as a particular detailed circumstance of error that is selected by a student. We hypothesized that the system promotes clear understanding regarding their compile error learning, which leads to encourage more experiences of compilation error correction, as well as to reduce unarticulated anxiety for the compile error. The system was tried in a self-assessment context at an undergraduate introductory programming course for approximately 100 non-CS students. The results in the questionnaire showed that the students appreciated the system for a reflection of their process, and we succeeded in reducing unarticulated anxiety for students. The results indicate that self-assessment with CocoViewer enable to boost students' motivation in programming education, which forms the basis of computer engineering education.

Keywords: computer engineering education; programming education; compile error correction; self-assessment; learning analytics

1. Introduction

In introductory programming environments using common text-based languages such as Java, C, or Python, learning to correct compile errors has been the first obstacle for novices. Seeing numerous compile errors has caused "nightmares" for many beginning programmers. In the classroom, teaching assistants are always busy supporting learners to correct errors. This results in both students and assistants having an inability to focus on the most important practice in the introductory level: thinking about workable algorithms. This had been manageable for CS (Computer Science) students, however, recent demands to develop higher level problem-solving skills using the computer, Computational Thinking [1], has broadened the target to non-CS students.

Non-CS students are feeling more negative regarding compile errors than teachers think. Our survey for students in our introductory programming course for non-CS students showed that 62% of students answered that they were feeling literal "fear" regarding compile errors. They also answered the question "What was the percentage of time you spent on compile error correction per total programming process?". The result of the students' perception was 28.4% where as our actual calculation was 15%.

In literature, researchers have discussed two approaches to solve the problem. One is to avoid compile error occurrences by using a visual programming language such as Squeak [2] or Scratch [3]. The visual programming approach enables beginning learners to build their program by block building as in a jigsaw puzzle style [4], which can completely eliminate any compile errors, because it is not possible to connect blocks unless those blocks can connect grammatically.

Another approach is to develop an environment which provides some scaffolding where the students can easily learn how to correct compile errors. Many students are worried about the number of occurrences for compile errors, however it is not a concern for professionals. The professionals can correct errors quickly because compile error messages give them clear clues for correcting errors. An advantage of this approach is that correcting error experiences promotes understanding of grammatical knowledge. This is a reasonable approach for redesigning compile error messages for novices, because the standard error messages that were assumed for professional use in design include many terminologies, therefore, it is hard to understand for beginners who have little grammatical knowledge. We can take the first approach described above (visual language) in the early stage of programming education.

However, it may be disappointing for highly motivated students, because many popular lan-

* Accepted 12 February 2016.

guages which provide attractive functions to solve problems by a computational approach, such as JavaScript in the browser, are still text-based. At least in a migration phase from visual to text-based language, they cannot avoid learning some grammatical knowledge. On the other hand, the second approach (error correcting scaffolding) never succeeded in completely swiping beginners' fear out. Once a student faces stacking in a number of compile errors, his/her motivation for learning programming is going to drop and is unlikely to return due to the negative experience.

In this paper, we tried to improve the situation in the second approach (error correcting scaffolding), by proposing a tool to encourage formative assessment regarding compile error correction learning by students themselves. Using the latest information technology, we can easily collect information concerning the learning procedure, and the tool providing visual information for students. Therefore, we took an approach using the computer as a learning-reflection tool [5]. Our hypothesis behind this is that the students' "fear" about compile errors came from their lack of information and knowledge regarding actual compile error learning status. Thus, from this point on in this paper we refer to "fear" as "unarticulated anxiety".

We hypothesized that the system promotes clear understanding regarding their compilation error learning, and the following three effects were expected:

(1) to encourage more experiences of compilation error correcting;
(2) reducing unarticulated anxiety concerning compilation errors;
(3) improvement of the procedure of correcting errors.

The remainder of this paper is organized as follows: We will discuss works related to this research in Section 2. We will introduce our course design, analysis tool in Section 3. Evaluation method is in Section 4. The results of the qualitative analyses are given in Section 5. Section 6 concludes with a discussion of the results and limitations of the study.

2. Related work

This study was built on research regarding individual skill growth in software engineering. Specifically, we focus on the chronological changes of the compile error correction time. PSP: Personal Software Process [6] is the classic theory and the only one which provides results of the chronological analysis of compile error correction time. Although the data was gathered in the professional development situation, PSP research results show the compile error correction time gradually decreases by experience. We can say it is an improvement of their skills and that seeing the phenomena visually encourages students toward further learning of programming. However, the original PSP has to be done by manual logging, which was unreasonable to conduct in the introductory programming course.

Recent information technology allows us to easily collect logs which record all operations in the students' computers. In this decade, many recording environments in the educational field have been proposed, resulting in a significantly low cost for collecting logs even in real-time (e.g. [7]). However, there are a few articles that try either analysis or visualization of the records, as we review in the following.

There are studies that attempted to discover indicators which show learning status by analyzing compile error correction logs. A number of reports statistically analyzed an occurrence of a compile error number in their programming education [8–10]. They examined the difficulty of each compile error kind by counting the occurrences. Jadud [8] examined calculating average correction time for each kind of detection of difficulty, and Thompson [11] concurred. Although the research concluded that the average time was useful to detect difficulties of the compile error kind, their research omitted the aspects of individual differences and temporal aspects of learning. Jadud [12] also tested a similar problem to detect the difficulty of each compile error kind. They proposed EQ (Error Quotient), which is calculated by how many times the students tried to compile in order to correct an error. However, the research omitted the aspects of correction time.

Recently, an approach in which manual qualitative analysis was included in the quantitative collected data was done in several studies. Marceau et al. [13] attempted to detect compile error difficulty for each kind by learners' activities. They defined activities which can be identified by learners' error correction processes, and their data in the actual education process was manually analyzed by the defined activities. Bringula et al. [14] proposed taxonomy of compile errors from a human cognitive aspect. The category was composed of six types (Thought error, Sensorimotor error, Omission Error, Memory error, Knowledge error, Habit error), and their regression analysis revealed the relationship between those types.

There is another approach for improving the problem by proposing software tools which directly support learners in solving their compile errors. In this category, the most frequent approach is to redesign compile error messages for novices [13, 15, 16]. Specifically, two reports show clear results were achieved through certain levels of improvement in the actual environment [13, 16]. Some research proposed tools that provide hints for learners to resolve their errors by using a database where the cases accumulated [17, 18]. The approach is similar to the expert system. A situation of programming language might also be affected by a language kind. In this view, some researchers claimed advantages of the particular language (e.g. Scheme [19]).

Our research does not take an approach which directly facilitates learners solving the problems they face, but takes the route which encourages the learners' learning process in the meta-cognitive layer. In this view, some studies tried a similar approach. For example, Chiken et al. [20] proposed a tool which supports the learners' reflection process using failure knowledge. In another example, Kay et al. [21] tried to let learners have self-assessment experiences for the activity of evaluation of several other's programs. Belski [22] also claimed the importance of the metacognitive process in programming education, although their research continues to conduct self-assessment by questionnaire.

3. CocoViewer

3.1 Architecture

We propose CocoViewer (Compile Error Collection Viewer) the name of which was from the system function characteristics: collection of compile error correction. The system visualizes compile error correction history for each learner in order to facilitate their analysis of their learning regarding compile error corrections and grammatical form of programming language. Knowing the accurate number and time spend for compile error correction removes learners' unarticulated anxiety regarding errors and correction.

We developed a system to support the learning of Java programming language. The system's whole dataflow structure is shown in Fig. 1.

The sensor which collects all activities' log including compile error occurrences/corrections is embedded in the Programming Development Environment used by learners in our educational situation. CocoViewer visualizes the logs which are accumulated in the working log database. Hence, all procedures from logging to visualization are completed automatically for uses. Although the system works on the standalone computer, the system is workable for analyzing all learners' logs in a class by connecting the Working Log Database via a server. The server system is not described in this paper as we focus on the usage of the personal reflection system.

3.2 Correction time calculation and chart

CocoViewer calculates a compile error correction time by activity logs for each compilation error occurrence. The difference in time between the error occurred and resolved, will be calculated in a general case. However, sometimes multiple errors occur or are resolved at the same time. For such a case, the system calculates by the amount of time spent in error collection BY the number of errors. An assumption of the method is the difficulty of corrections is equal in every case. Although it is not always accurate, we had to take this method for the sake of automatic calculation.

Using a correction time history, CocoViewer generates a correction time chart. A chart is created for each compile error kind. An example of the chart is shown in Fig. 2. The figure is an example for the compile error kind of "reached end of file while parsing" in Java. X-axis of the chart shows the compile error correction opportunity's number (unit: the sequence number). The number of 1 means the first opportunity to resolve this kind of error, and the maximum number (19 in this example) means the last opportunity for the error. Y-axis of the chart shows the error correction time (unit: seconds). For example, we can read in Fig. 2 that the learner spent a compile error correction time of 65 seconds the first time and 34 seconds the second

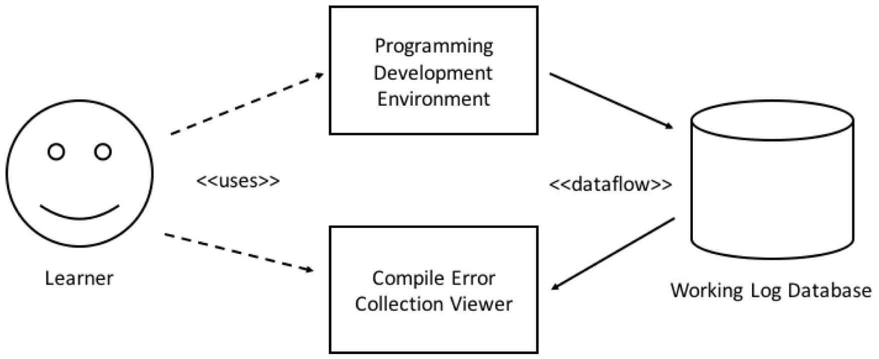

Programming Development Environment
<<uses>>
<<dataflow>>
Learner
Compile Error Collection Viewer
Working Log Database

Fig. 1. Different approaches in assessment for programming assignments.

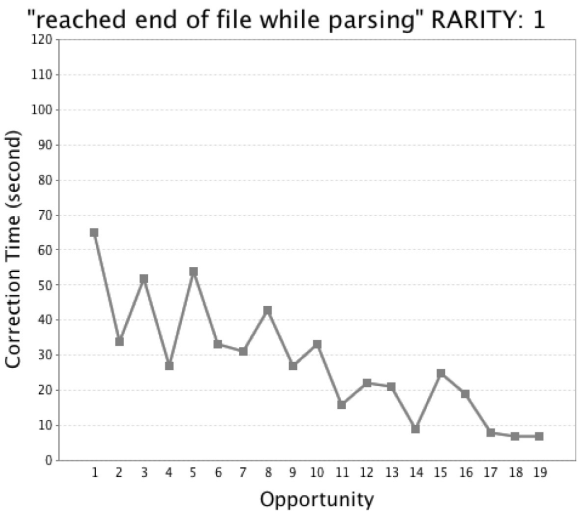


Fig. 2. An example of a correction time chart.

time. Hence, the learner can see the trajectory of their error correction time history, which is expected to be improved by additional experiences.

3.3 CocoViewer dashboard

CocoViewer has a dashboard window to provide at a glance view of personal compile error correction information. An example of a dashboard is shown in Fig. 3.

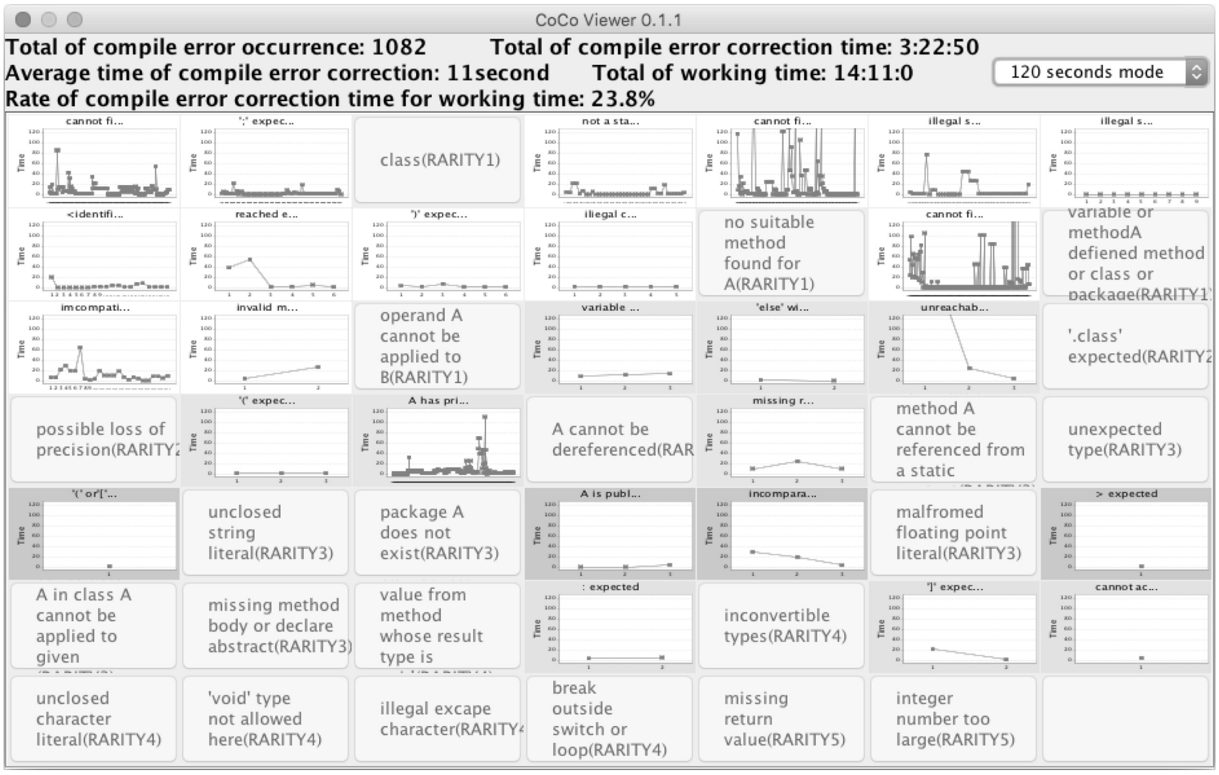
A dashboard is comprised of two major compartments. One is the upper compartment which provides summary information regarding the personal compile error correction history. The summary includes the following information:

- Total of compile error occurrence: (ex: 1092).
- Average time of compile error correction: (ex: 11 seconds).
- Total of compile error correction time: (ex: 3:27:10 hh:mm:ss).
- Total of working time: (ex: 14:11:00 hh:mm:ss).
- Rate of compile error correction time for working time: (ex: 24.3%).

Each example shown above is the actual student's data in our introductory programming course, which was carried out over 15 weeks. As this example indicates, there is important information for learners to know the current status toward an accurate analysis.

The bottom compartment of a dashboard shows the tile representation of correction time charts for all kinds of compile errors. There are 7 × 7 tiles, which can show the maximum of 49 kinds of error correction time charts. In default, each of 48 Java compile error kind is assigned to one tile, ordered by a frequency profile for each kind of error from top left to bottom right. The frequency profile used for this study was created by the statistics for all of students' logs collected in the introductory class carried out the year before this study.

The dashboard is designed for promoting "col-

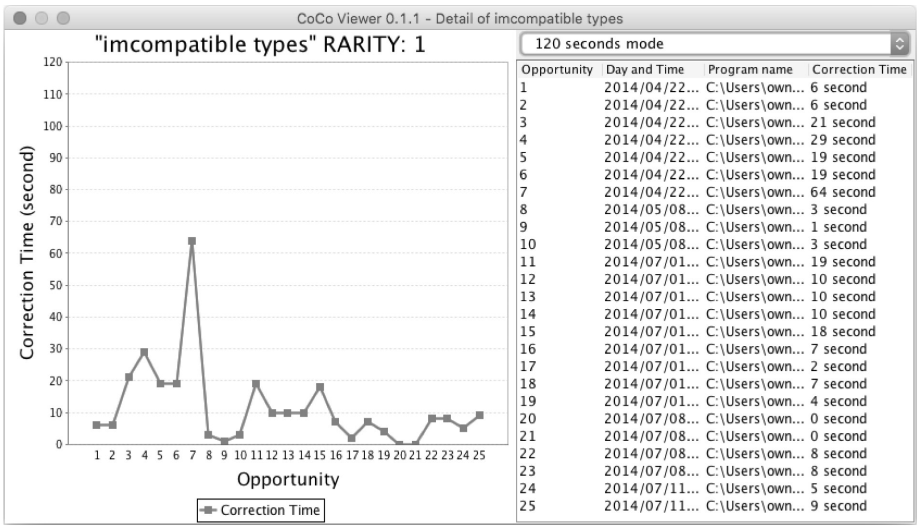

CoCo Viewer 0.1.1

Total of compile error occurrence: 1082 Total of compile error correction time: 3:22:50
Average time of compile error correction: 11second Total of working time: 14:11:0
Rate of compile error correction time for working time: 23.8%

120 seconds mode

cannot fi...
';' expec...
class(RARITY1)
not a sta...
cannot fi...
illegal s...
illegal s...
<identifi...
reached e...
')' expec...
illegal c...
no suitable method found for A(RARITY1)
cannot fi...
variable or methodA defiened method or class or package(RARITY1
imcompati...
invalid m...
operand A cannot be applied to B(RARITY1)
variable ...
'else' wi...
unreachab...
'.class' expected(RARITY2
possible loss of precision(RARITY2
'(' expec...
A has pri...
A cannot be dereferenced(RAR
missing r...
method A cannot be referenced from a static
unexpected type(RARITY3)
'(' or'[' ...
unclosed string literal(RARITY3)
package A does not exist(RARITY3)
A is publ...
incompara...
malfromed floating point literal(RARITY3)
> expected
A in class A cannot be applied to given
missing method body or declare abstract(RARITY3)
value from method whose result type is
: expected
inconvertible types(RARITY4)
']' expec...
cannot ac...
unclosed character literal(RARITY4)
'void' type not allowed here(RARITY4)
illegal excape character(RARITY4
break outside switch or loop(RARITY4)
missing return value(RARITY5)
integer number too large(RARITY5)
Time

Fig. 3. The CocoViewer dashboard.


CoCo Viewer 0.1.1 - Detail of imcompatible types
"imcompatible types" RARITY: 1
Correction Time (second)
Opportunity
Correction Time

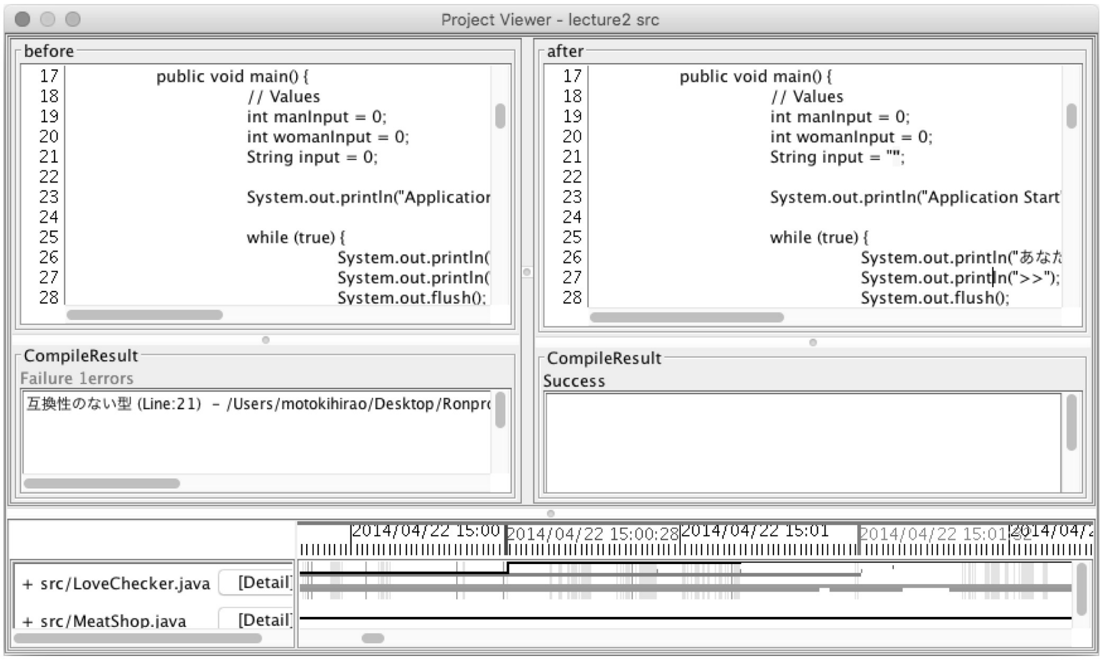
120 seconds mode

Opportunity	Day and Time	Program name	Correction Time
1	2014/04/22...	C:\Users\own...	6 second
2	2014/04/22...	C:\Users\own...	6 second
3	2014/04/22...	C:\Users\own...	21 second
4	2014/04/22...	C:\Users\own...	29 second
5	2014/04/22...	C:\Users\own...	19 second
6	2014/04/22...	C:\Users\own...	19 second
7	2014/04/22...	C:\Users\own...	64 second
8	2014/05/08...	C:\Users\own...	3 second
9	2014/05/08...	C:\Users\own...	1 second
10	2014/05/08...	C:\Users\own...	3 second
11	2014/07/01...	C:\Users\own...	19 second
12	2014/07/01...	C:\Users\own...	10 second
13	2014/07/01...	C:\Users\own...	10 second
14	2014/07/01...	C:\Users\own...	10 second
15	2014/07/01...	C:\Users\own...	18 second
16	2014/07/01...	C:\Users\own...	7 second
17	2014/07/01...	C:\Users\own...	2 second
18	2014/07/01...	C:\Users\own...	7 second
19	2014/07/01...	C:\Users\own...	4 second
20	2014/07/08...	C:\Users\own...	0 second
21	2014/07/08...	C:\Users\own...	0 second
22	2014/07/08...	C:\Users\own...	8 second
23	2014/07/08...	C:\Users\own...	8 second
24	2014/07/11...	C:\Users\own...	5 second
25	2014/07/11...	C:\Users\own...	9 second

Fig. 4. Correction time chart window.

lection'' of compile errors for learners. For each tile, a thumbnail of a correction time chart will be shown if there is at least one compile error correction record for the kind of error assigned in the tile. Otherwise the cell is colored in gray with the name of the kind of error. The design of fixed location for each error is essential in order to give learners accurate information for their experiences. All the blank (in gray) tiles in the initial state, are expected to be filled by experiences. From the aspect of ''collection'' in this view, an occurrence of compile error is no longer a negative for novice learners. A fluency of fixing compile error and understanding of grammar has to be correlated with experiences. Hence, the system encourages learners to enrich experiences of any kinds of compile errors.

Moreover, there is one more designed factor for letting learners motivate the ''collection''. RARITY is assigned to every kind of compile error to guide the difficulty level of the collection/correction of each compile error. The range of 1 to 6 of RARITY was assigned based on the frequency profile. The background color of a tile shows the RARITY of the error. For example, white for RARITY 1, green for RARITY 2, or pink for RARITY 3 is used for the background color of tile respectively. RARITY guides not only the level of difficulty for the compile error, but also



before:

```
        public void main() {
                // Values
                int manInput = 0;
                int womanInput = 0;
                String input = 0;

                System.out.println("Applicatior

                while (true) {
                        System.out.println(
                        System.out.println(
                        System.out.flush();
```

after:

```
        public void main() {
                // Values
                int manInput = 0;
                int womanInput = 0;
                String input = "";

                System.out.println("Application Start

                while (true) {
                        System.out.println("あなた
                        System.out.println(">>");
                        System.out.flush();
```

CompileResult: Failure 1errors

```
互換性のない型 (Line:21) - /Users/motokihirao/Desktop/Ronpr
```

CompileResult: Success

2014/04/22 15:00 | 2014/04/22 15:00:28 | 2014/04/22 15:01 | 2014/04/22 15:01:30 | 2014/04/2

+ src/LoveChecker.java

+ src/MeatShop.java

Fig. 5. Correction details window.

encourages the collection of rare items to fill the collection case.

3.4 Correction time chart and details window

By clicking one tile of the CocoViewer dashboard, users can open a correction time chart window. An example of the window is shown in Fig. 4. This example is regarding the kind of "incompatible types". A correction time chart window includes a correction time chart and a correction history table. The table shows details of each record, such as the date, the target program, and the correction time.

Users can explore further details of the situation for the error correction opportunity, by clicking one record of histories on the table. An example of a correction details window is shown in Fig. 5. The window visualizes the two source code comparisons: the left pane shows the source code when the error happened, and the right pane shows the source code when the error resolved. In addition, users can use the time slider to move the time and check the source code at the time, where the functionality is implemented by the Programming Process Visualizer [23]. Hence, users can check what was happening in the situation of the error and how to solve the compile error.

3.5 Usecase: How learners analyze their charts

We designed CocoViewer for learners in order that they explore their learning curve by analyzing the shape of their correction time chart. From our experiences, the shape can be classified by 3 kinds of patterns: (a) Decreasing, (b) Jagged, and (c) Increasing. Examples of these patterns are shown in Fig. 6.

Fig. 5(a) shows an example of the decreasing type of correction time chart. The type of chart clearly shows that the learner is gradually coming to understand through the experiences of compile error correction. It is the ideal for a dashboard to be filled with this type of a chart.

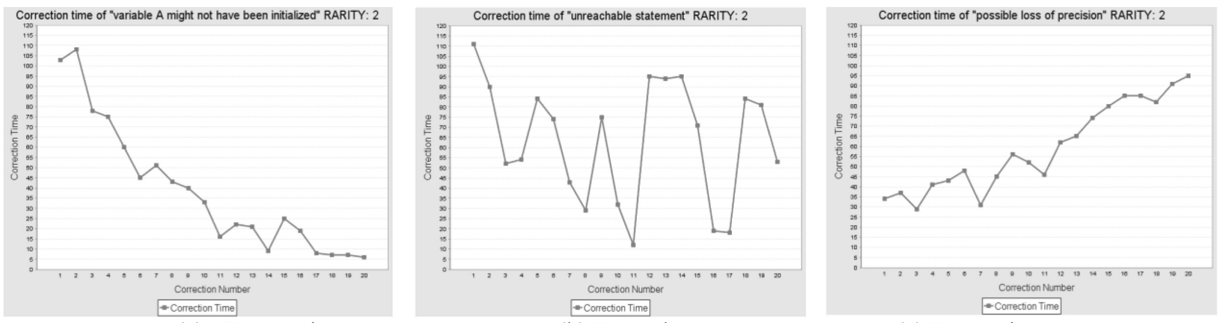
Fig. 5(b) shows an example of the jagged type of correction time chart. Several factors are considered for reasoning with this type. (1) Learners struggling with the acquisition of the grammatical rules are related to the compile error. It is not uncommon that the same error has appeared by different causes. Another reason could be that it is difficult to understand the compile error message. (2) There is a problem on the correction time calculation method; explained in section 3. 2, there is a possibility resulting in a big difference in the calculated correction time, depending on the situation of whether single error correction or multiple errors combined has occurred. (3) There is a problem in the learners' internal procedure to learn by experience. In the actual classroom, teachers typically observe students who are trying to fix compilation errors without any thought for the reasoning behind the errors. Otherwise, (4) it falls within the procedure of understanding error correction. In this case, the type of chart is expected to shift to the pattern (a). A qualitative analysis using the correction details window is necessary in order to detect either of one or more reasons, in any case.

Fig. 5(c) shows an example of the increasing type of the correction time chart. Although it should not be expected to appear, it is not uncommon to see some in an individual dashboard. The same reasons for Jagged types are considered as the causes. In typical instructional design for introductory programming guides learners proceed from easier, simpler tasks to more difficult, complex ones. This means the situation of the task (difficulty of the whole source code) might be a tough situation even if the same type of the compile error has appeared. Further qualitative analysis is necessary in this case, as well as in case of the pattern (b).

4. Experimental study

4.1 Research question and hypotheses

We conducted an empirical study in our introductory programming class. The goal of the study was to evaluate effectiveness of the proposed environment (CocoViewer). The research question was



(a) Decreasing (b) Jagged (c) Increasing

Fig. 6. Patterns of correction time chart.

"How CocoViewer can be used in the classroom for learners as a self-assessment tool of their compile error correction process".

Students were given the opportunity to use CocoViewer, and analyze their own data for a self-assessment of their learning procedure. The experimental session was held at the 12th of 15 weeks of the course; therefore they could use data that have been recorded over 11 weeks during their coursework for all assignments. We began with two hypotheses:

Hypothesis I. Learners can reduce "unarticulated anxiety" by the following processes

(1) View correction time charts, and analyze the charts' shape.
(2) Decrease the gap between their expectation and actual for compile error correction profile.
(3) Recognize extent of learning for their compile error correction.

Hypothesis II. The activity of self-assessment with viewing the visualization of their own procedural data, is interesting itself for many students. They can enjoy the activity, start to explore their own data driven by intrinsic motivation, and find the way to improve their learning process.

4.2 Educational environment descriptions

The introductory programming course was designed for art students, rather than for computer science students. Therefore, the objective of the course was to develop an understanding of task-oriented programming. The objective was independent from any programming language, although Java language was used for the actual environment. Approximately 100 students participated in this course; two lecturers and six teaching assistants conducted the class.

4.3 Procedure

We performed our experiment in the 12th of 15 weeks of the class. Students participated in this experiment in a semi-voluntary situation. Although the experiment was conducted as a part of the class, the task was not mandatory: if learners thought the experience of CocoViewer was not useful, they could decide not to join the experiment without any penalties in the grading.

The experimental study was conducted using a worksheet which includes the following contents.

1. Instructions how to operate and to launch and use CocoViewer.
2. Instructions of how to analyze a correction time chart, as it was explained in Section 3.5.
3. Activity1: analyzing my own dashboard, select a few correction time charts and analyze them.
4. Activity2: Discussing with others the results of activity1 with the comparison of others' dashboard.
5. Questionnaire: questions regarding the verification of the Hypothesis I and II as described in Section 5.

The total time the learners used in this task was approximately 40 minutes. At first, 10 minute instruction was done by a teacher using 1 and 2 of the worksheet. After that, students could freely proceed in the experiment using the worksheet. In our observation, it took 10–20 minutes for Activity 1 and 2, and 10 minutes for marking the questionnaire.

We received 71 sheets of questionnaires out of 100 registered students in the class. This means 71% of students selected to be a volunteer of this experiment. However, as 10 sheets of the questionnaire were not completed, we used 61 sheets for analysis in this paper.

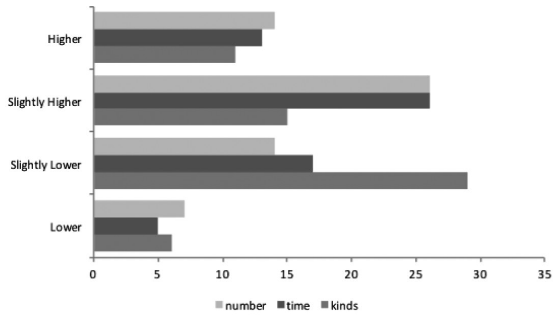
5. Results

5.1 Results for comparison between expected and actual and reducing fear

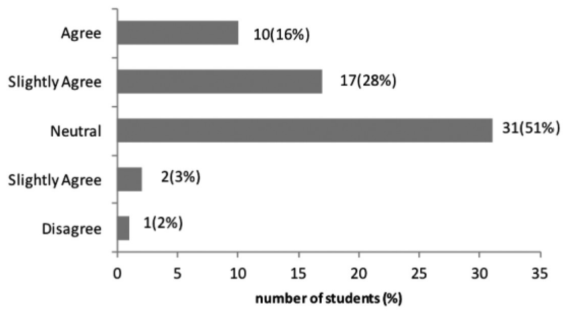
The results of questionnaire are shown in Fig. 7. The responses for the question "What is the difference between expected and actual for the compile error profile?" are shown in Fig. 7(Q1). We asked for 3 factors: the total of error occurrences, the number of kinds of errors, and the total of compile error correction time. Students answered via four scales of higher number/lower number for occurrences and kinds, longer/shorter for times respectively.

The figure shows that 66% of students thought the number of the total of error occurrences was higher than expected. A similar result is shown for the correction time: 64% of the students thought the actual correction time was longer than they expected. The percentage indicates approximately twice of the students answered higher/longer side. An opposite result can be observed for kinds: 58% students thought that the number of kinds was lower than they expected.

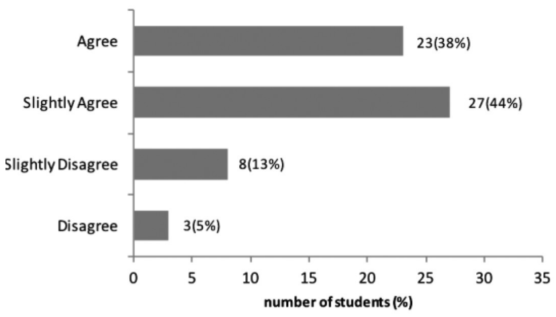
The responses for the question "Was CocoViewer useful to reduce your fear against compile errors?" are shown in Fig. 7(Q2). The question was aimed at acquiring direct responses regarding the effect of reducing "unarticulated anxiety". As a result, 44% of the students marked "agree" and "slightly agree", and almost all others marked "neutral". As our preliminary survey showed that 62% of students answered that they were feeling "fear" regarding compile errors, the result can be evaluated as positive.



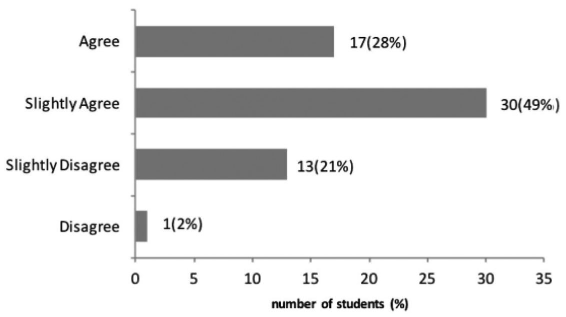
(Q1) What is the difference between expected and actual for the compile error profile.



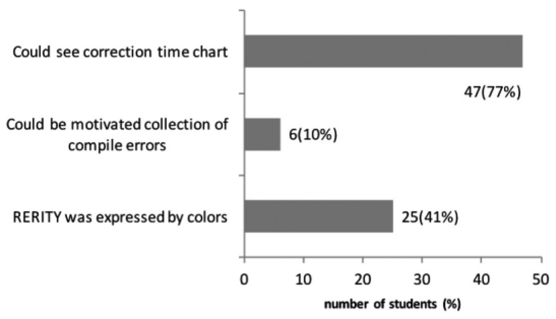
(Q2) Was CocoViewer useful to reduce your fear against compile errors?



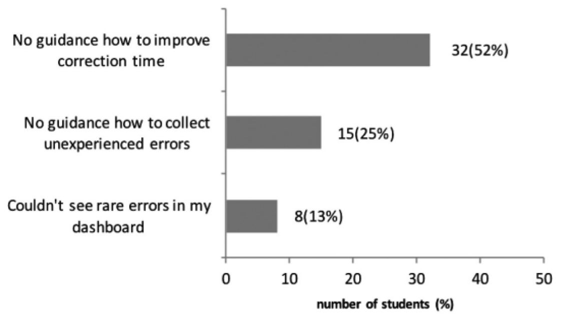
(Q3) Was it an interesting experience to see your own compile error correction history?



(Q4) Is CocoViewer useful to learn compile error correction?



(Q5) What was the interesting feature?



(Q6) What was the frustrating feature?

Fig. 7. Results of questionnaire.

5.2 Results for general reaction, usefulness of CocoViewer

The responses for the question "Was it an interesting experience to see your own compile error correction history?" are shown in Fig. 7(Q3). 82% of the students marked "interesting". The responses for the question "Is CocoViewer useful to learn compile error correction?" are shown in Fig. 7(Q4). 77% of the students marked it was "useful". These results indicate that students generally accepted the CocoViewer experience, with understanding of the value for their learning. In addition, as the Japanese word for "interesting" includes a meaning of "I enjoyed", the result also indicates that students could enjoy the CocoViewer experience; as in exploring their own learning history.

In order to illustrate advantages and disadvantages of CocoViewer for users, there were two questions in the questionnaire. The responses for the question "What was the interesting feature?" are shown in Fig. 7(Q5). 77% of students marked "Could see correction time chart", which means students were encouraged by checking their reduction of correction time. Nine percent of the students marked the "Could be motivated collection of compile errors", meaning some of the students were motivated by the "collection" factor. Forty one percent of the students marked "RARITY was

Table 1. High quality surveys of automatic testing systems

Student ID	Comments
S1	My fear against compile error was dramatically reduced after going through this worksheet using CocoViewer.
S2	I found in my chart that the correction time is reducing after the second opportunity.
S3	My chart clearly showed that the correction time was reducing, though it is obvious.
S4	I found that my correction abilities are different depending on the kind.
S5	There were few reducing type chart in my dashboard. I clearly understood that I have not got used to fixing errors.
S6	Most of my charts were jugging.
S7	I enjoyed the worksheet. It was a good opportunity to reflect my compile error correction.

expressed by colors'', which indicates that the idea of colored RARITY worked well for students to know the levels of compile errors.

The responses for the question ''What was the frustrating feature?'' are shown in Fig. 7(Q6). Fifty two percent of the students marked ''No guidance how to improve correction time''. Twenty four percent of the students marked ''No guidance how to collect unexperienced errors''. These results indicate that the students were motivated for further learning after understanding their current learning status. The tool has to support the learners' further exploration for improving their learning process. Thirteen percent of the students marked ''Couldn't see rare errors in my dashboard''. This reinforced the results that some of the students are motivated by the ''collection'' factor.

5.3 Comments

Typical comments written in the questionnaire were chosen by the authors to complement the results of the students' reactions from using CocoViewer. These are shown in Table 1.

The comment of S1 is a typical example of the student describing the reduction of fear through the experience. S2, S3, and S4 described the typical positive reaction by seeing the reducing type of correction time chart. The comments of S5 and S6 include some negative information regarding a result of analysis for shapes of charts. However, they could accurately understand their current status, and we did not find comments expressing emotionally negative feelings as a result. The comment from S7 is also a typical one in that an acceptable and an enjoyable feeling for the experiment was described.

6. Discussion

6.1 Evaluation of H1. Reducing ''unarticulated anxiety'' by closing the gap between expected and actual

The primary method for reducing ''unarticulated anxiety'' was that the learner can close the gap between a vague image of compile error correction and the actual method of compile error correction. The result of exploring the gap between their expectation and actual for the learning data revealed the following:

- The actual number of error occurrences was HIGHER
- The actual time they used for correction was LONGER
- The actual number of error kind they experienced was LOWER than their expectation, respectively.

We can interpret the learners were encouraged by this result, understanding they actually have richer experiences than they thought. However, there is a possibility that the learners were disappointed by these results as they might not feel their learning outcome reached their expected level, considering their actual effort. We could not find any comments which indicated that feeling. Additionally, the other comments which supported the learners were encouraged by seeing the shapes of correction time charts.

In addition, it is especially important that learners experienced few compile error kinds; the occurrence of compile error kinds was minimal. This means learners realized that they primarily had been correcting similar compile errors. The effect of this is considered that learners can be encouraged by knowing they are producing sufficient effort,, as well as they can know they only have to master a tiny set of correction knowledge.

There were still some unexpected learners who did not experience a decreasing type of chart. In such a case, we could not conclude they succeeded in reducing their ''unarticulated anxiety''. How to support such a student should be considered at the next step of this research.

6.2 Evaluation of H2. Acceptable as self-assessment tool, used by intrinsic motivation

First of all, we evaluated the result that 71% of the students decided to use CocoViewer, despite the fact that they were not given a direct merit of grading. The result shown at Section 5.2 indicates that the majority of learners could enjoy the experiment itself, and they appreciated the value of their learning compile error correction. The result shown at

Section 5.1 clearly indicates they succeeded to engage in the quantitative inspections. In addition, the results shown in section 5.3, students such as S2 to S6 described the results of their qualitative analysis for their chart shapes. By seeing these results, we can consider that the learners' activity was generally driven by intrinsic motivation. Accordingly, we concluded that the contribution of CocoViewer gave them opportunities to think regarding their own learning, along with having fun.

What we revealed in this experimental study was limited in giving the opportunity for learners to start thinking about their learning process. As we can see in the results in Section 5.2, students require the tools for the next step for improving their learning. Further research will be needed for both the tool assists to help improvement of procedures; the verification of the procedure for correcting errors has certainly improved.

7. Conclusions

We developed a system that allows learners to analyze their compile error correction records. Our experiment took place in a novice programming class with approximately 100 students. Using the system increased learner interest in compile error correction study, and, at the same time, learners indicated an interest in using the system. The system succeeded in helping reduce "unarticulated anxiety" for compile error by learners by closing the gap between the learner's compile error expectation and actual correction experience. We believe that the results indicate that self-assessment with CocoViewer succeeded to boost students' motivation in programming education, which has to form the robust basis of computer engineering education.

References

1. J. M. Wing, Computational Thinking, *Commun. ACM*, **49**(3), pp. 33–35.
2. D. Ingalls, T. Keahler, J. Maloney, C. Wallace and A. Key, Back to the Future: The Story of Squeak. A Practical Smalltalk Written in Itself, *Proc. of ACM OOPSLA '97*, **32**(10), 1997, pp. 318–326.
3. Scratch Team Lifelong Kindergarten Group MIT Media Lab, (n.d.). Scratch—Imagine, Program, Share. Accessed February 21, 2015 from http://scratch.mit.edu/
4. J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman and M. Resnick, Scratch: a sneak preview. *Proceedings Second International Conference on Creating Connecting and Collaborating through Computing*, 2004, pp. 104–109.
5. A. Collins and J. S. Brown, The computer as a tool for learning through reflection, In: Mandl, H. and Lesgold, A. (Eds.): *Learning for Intelligent Tutoring Systems*, 1988, pp. 1–18, Springer-Verlag, New York.
6. S. W. Humphrey, Introduction to the Personal Software Process, *SEI Series in Software Engineering*, Addison-Wesley, 1997.
7. A. Alammary, A. Carbone and J. Sheard, Implementation of a Smart Lab for Teachers of Novice Programmers, *Proceedings of the Fourteenth Australasian Computing Education Conference*, **123**, 2012, pp. 121–130.
8. M. C. Jadud, A first look at novice compilation behaviour using BlueJ, *Computer Science Education*, **15**, 2005, pp. 25–40.
9. J. Jackson, M. Cobb and C. Carver, Identifying Top Java Errors for Novice Programmers, *Frontiers in Education*, 2005. FIE '05. *Proceedings of the 35th Annual Conference, T4C24-T4C27*
10. C. T. I. Mow, Analyses of Student Programming Errors In Java Programming Courses, *Journal of Emerging Trends in Computing and Information Sciences*, **3**(5), 2012, pp. 739–749.
11. S. M. Thompson, An Exploratory Study of Novice Programming Experiences and Errors. Master's thesis, *University of Victoria*, 2006.
12. M. C. Jadud, Methods and tools for exploring novice compilation behaviour. In: *Proceedings of the second international workshop on Computing education research*, 2006, pp. 73–84.
13. G. Marceau, K. Fisler and S. Krishnamurthi, Mind your language: on novices' interactions with error messages. In: *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2011, pp. 3–18.
14. R. Bringula, G. M. Manabat, M. A. Tolentino and E. Torres, Predictors of Errors of Novice Java Programmers, *World Journal of Education*, **2**(1), 2012, p. 3.
15. M. Hristova, A. Misra, M. Rutter and R. Mercuri, Identifying and correcting java programming errors for introductory computer science students, *ACM SIGCSE Bulletin*, **35**(1), 2003, pp. 153–156.
16. M. H. Nienaltowski, M. Pedroni and B. Meyer, Compiler Error Messages: What Can Help Novices? In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, **40**(1), 2008, pp. 168–172.
17. C. Watson, F. W. Li and J. L. Godwin, Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. *Advances in Web-Based Learning—ICWL*, 2012, pp. 228–239.
18. B. Hartmann, D. MacDougall, J. Brandt and S. R. Klemmer, What Would Other Programmers Do: Suggesting Solutions to Error Messages, In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 2010, pp. 1019–1028.
19. M. Crestani and M. Sperber, Experience report: growing programming languages for beginning students, In: *Proc. 15th ICFP*, **45**(9), 2010, pp. 229–234.
20. K. Chiken, A. Hazeyama and Y. Miyadera, A Programming Learning Environment Focusing on Failure Knowledge, *J. IEICE in Japan*, J88-D1 (1), 2005, pp. 66–75.
21. J. Kay, L. Li and A. Fekete, Leaner Reflection in Student Self-assessment. *ACE '07 Proceedings of the Ninth Australasian conference on Computing education,* **66**, 2007, pp. 89–95.
22. I. Belski, The Impact of Self-assessment and Reflection on Student Learning Outcomes, In: *Proceedings of the 21st Annual Conference for the Australasian Association for Engineering Education*, 2010, pp. 216–221.
23. Y. Matsuzawa, K. Okada and S. Sakai, Programming Process Visualizer: A Proposal of the Tool for Students to Observe Their Programming Process. In: *Innovation and Technology in Computer Science Education (ITiCSE '13)*, 2013, pp. 46–51.

Yoshiaki Matsuzawa is an assistant professor at Graduate School of Informatics, Shizuoka University, Japan. He received his Ph.D. in Media and Governance in 2008 from Keio University, Japan. His research interests include software-engineering education, educational software development, modeling and simulation of complex systems, and learning sciences.

Motoki Hirao is a graduate student at Graduate School of Informatics, Shizuoka University, Japan. He received a bachelor degree in 2014 from Shizuoka University, Japan. His research interest includes educational software development.

Sanshiro Sakai is a professor at Graduate School of Informatics, Shizuoka University, Japan. He received his Ph.D. in engineering in 1984 from Shizuoka University, Japan. He is. His research interests include computer supported collaborative learning, software development environment and programming learning support system.