Applying Gamification to Motivate Students to Write High-Quality Code in Programming Assignments

Remin Kasahara
Waseda University
Tokyo, Japan
remin@ruri.waseda.jp

Kazunori Sakamoto
Waseda University
Tokyo, Japan

Hironori Washizaki
Waseda University
Tokyo, Japan

Yoshiaki Fukazawa
Waseda University
Tokyo, Japan

ABSTRACT

Background: Traditional programming education focuses on training students' ability to write correct code that meets the specifications in programming assignments. In addition to correctness, software engineering studies argue that code quality is important. **Problem:** Nurturing students' ability to write high-quality code in programming assignments is difficult due to two main reasons. (1) Considering code quality while grading is undesirable because there are no objective and fair measurement metrics. (2) Grading assignments from multiple viewpoints (correctness and quality) is difficult and time-consuming. **Approach:** We propose applying gamification with code metrics to measure code quality in programming assignments. Our approach can motivate students to write code with good metric scores independent of grading. We implemented our approach and conducted a control experiment in a programming course at a university. **Result:** Our approach did not interfere with students' submissions but improved metric scores significantly. Hence, our approach can engage students to write high-quality code.

CCS CONCEPTS

• **Applied computing** → **Education**; • **Social and professional topics** → *Software engineering education*; • **Human-centered computing**;

KEYWORDS

programming education; code quality; code metrics; leaderboard; gamification; online judge

ACM Reference Format:
Remin Kasahara, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. 2019. Applying Gamification to Motivate Students to Write High-Quality Code in Programming Assignments. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland, UK*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304221.3319792

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland, UK

ACM ISBN 978-1-4503-6301-3/19/07.
https://doi.org/10.1145/3304221.3319792

1 INTRODUCTION

It is important to foster students' ability to write a program that meets specifications. In programming education, most assignments and examinations focus on writing programs that meet specifications (i.e., writing correct programs). For example, an online judge, which is an automatic programming assignment grading system [13, 15], judges whether the submitted program meets the specifications. Teachers usually motivate students to submit correct programs for assignments by grading them because meeting specifications is a clear and fair grading criterion.

It is also important to nurture students' ability to write high-quality source code because low-quality code leads to problems such as reduced productivity [6]. To develop ability, students need to be motivated to write high-quality code in programming assignments. Although grading is a good way to motivate students, it is difficult to grade assignments with regard to code quality for two main reasons. (1) Since objective and fair metrics do not exist, grading code quality is undesirable. (2) Grading assignments comprehensively from multiple viewpoints (correctness and quality) is difficult and time-consuming.

To overcome these problems, herein we propose a combination of gamification and code metrics, to motivate students to write high-quality code in assignments. Gamification can raise student motivation and stimulate certain actions [9]. Code metrics (e.g., lines of code (LOC) and cyclomatic complexity (CC) [17]) measure the quality of code and represent it as a score. Although a score does not necessarily indicate good code, code with a good score tends to be high-quality [6]. Since gamification does not require complete fairness, it can motivate students to improve metric scores, and may engage students to write high-quality code without grading.

Numerous studies have investigated how to successfully motivate students by gamification [4, 22], and several papers have reported the undesirable effects of gamification [27]. Herein, we focus on a leaderboard, one of the most major gamification elements to evaluate our approach systematically.

We set the following three research questions:

RQ1 Does an online judge with a leaderboard using the CC change the number of problems solved by each student?
RQ2 Does an online judge with a leaderboard using the CC change the CC of code written by students?
RQ3 Does an online judge with a leaderboard using the CC change the code quality?

Our research contributions include:

- We proposed a leaderboard using the CC in an online judge. This setup evaluates the quality of the submitted code and provides instant feedback about the correctness and code quality.
- We added a leaderboard to an existing online judge that students used in a programming course.
- We analyzed the results of multiple assignments in the experiment systematically by employing meta-analysis as well as the submitted code by hand.
- We found that the leaderboard improved with the CC scores with a statistical significance in the intervention (RQ2), and it did not interfere with students' submissions (RQ1).
- We found two cases where students improved code quality during the intervention and one case where a student stuck to reduce the CC scores without improving the code quality (RQ3).

2 BACKGROUND

The **online judge** is a web application that automatically grades programming assignments [15]. When students submit code to a programming problem to an online judge, it compiles the submitted code to generate and then test a program. The online judge tests the program by checking whether the program yields the expected outputs corresponding to the given inputs. Consequently, more than one program can be correct. The online judge not only helps students learn programming with its instant feedback but also assists teachers in monitoring students' learning. Several online judges have been proposed and developed for educational purposes [13, 21, 23]. Some organizations and companies use them today to manage online programming contests, e.g., Code Jam [1] and Codeforces [2].

In software engineering, several **code metrics** (e.g., LOC, CC, and Halstead Complexity [11]), and Chidamber and Kemerer metrics (CK metrics) [7] have been proposed to measure and evaluate code quality [18, 28]. Software developers use code metrics to improve code quality because extremely complicated code lowers productivity [6].

Deterding et al. defined **gamification** as the use of game design elements in non-game contexts [9]. They classified game design elements into five levels: game interface design patterns (e.g., points, badges, leaderboards, levels), game design patterns and mechanics (e.g., time constraints, limited resources, turns), game design principles and heuristics (e.g., enduring play, clear goals, variety of game styles), game models, and game design methods. The goal of gamification is to motivate users' activities.

3 PROPOSAL

To encourage students to write high-quality code, we employ gamification that shows code metrics as a score in an online judge. The target is students in an intermediate programming course.

3.1 Method

The following procedure was used in our online judge system, which was extended for gamification:

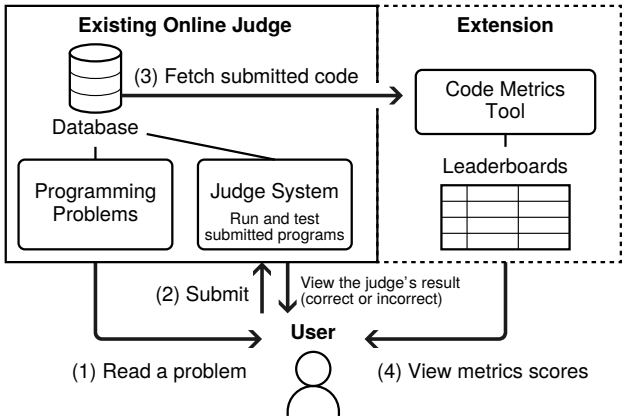


Figure 1: Implementation overview.

(1) A student answers a programming problem by submitting code to an online judge system.
(2) The system measures the CC of the submitted code.
(3) The system instantly shows the CC score with a leaderboard to the student.
(4) The student resubmits his/her code zero or more times, and the system updates the results. The system records the smallest CC score.

Step 1 is the same as the original system. The online judge asks students to solve competitive-programming-style tasks. For the second and third steps, the system provides the score to the students.

This paper selected CC as the score element due to its theoretical basis and usability [28]. The CC reflects the code complexity well compared to the file size or LOC. In addition, the concept of the CC is easy for students to comprehend and use. However, the system may use another metric and instructors can change the metrics according to their viewpoints.

The essential idea of this method is not the selection of code metrics but the competition by a leaderboard with code metrics measurement. We employed a leaderboard from gamification elements for two reasons. (i) Since it is difficult to select suitable threshold scores of code metrics to judge whether code is good or bad, students should be able to compare their score with others instead of a predetermined goal to evaluate their score. (ii) The leaderboard concept is familiar in competitive programming such as a contest. To ensure that this experiment is systematic and reproducible, we did not employ other gamification elements.

In step 4, we expect gamification to motivate students to write better code. Thus, neither the system nor the instructors forced the students to resubmit.

3.2 Implementation

Figure 1 overviews the implementation of the gamification system. We used the Waseda Online Judge (WOJ) as our online judging system. WOJ is a web application provided by the department of computer science at Waseda University. To implement gamification, we developed an extensional web application: WOJ Standings (WOJS). The WOJS shares a database with the online judge. When

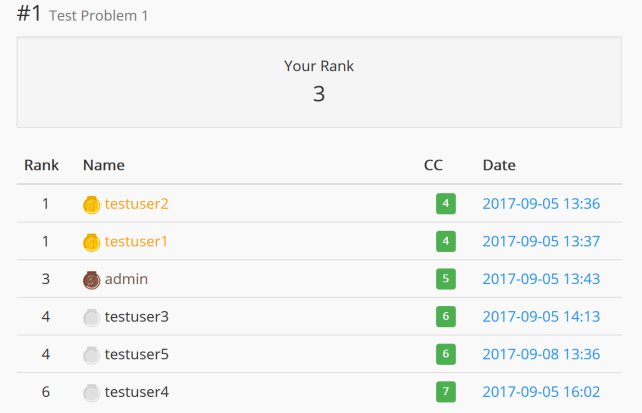


Figure 2: Screenshot of the leaderboard on the WOJS.

a student submits code to the online judge, the WOJS measures the CC and shows the student his or her score.

The CC is usually calculated for each module such as class and function. The code complexity analyzer we used in WOJS, Lizard [3], measures the CC for each function. In software development, code metrics tools monitor whether the CC exceeds the reference value. Developers are making efforts to avoid functions with a large CC. Thus WOJS used the largest CC of a program as the score.

Students can compare their scores to those of other students on the leaderboard (Fig. 2). The leaderboard shows ranks, user names, CC scores, and submission time. The leaderboard is generated for each programming problem because the CC values depend on the problem. Students can edit their own user name freely. The lower the CC score, the higher his/her rank. Users ranked from first to third are displayed in different colors.

4 EVALUATION

We conducted a six-week experiment using the WOJS in a C programming course to evaluate the effectiveness of our system.

4.1 Experiment

Participants were students taking a C programming course at Waseda University. Because all participants had completed an introductory programming course the previous year, they could solve primary programming problems independently. We provided students with an overview of the experiment and an explanation of CC during the first lecture. A total of 35 participants submitted a consent form. We randomly divided the participants into two groups: group A (17 students) and B (18 students). We also divided the course period into two halves: weeks 1 to 3 and weeks 4 to 6. For the control experiment, students in group A used the WOJS only in the first half, while group B used it only in the second half. Each week, all students in the course were asked to solve three programming problems, which were turned into the online judge. These problems had varying levels of difficulty. Problems 1 and 2 were appropriately difficult, whereas Problem 3 consisted of more challenging tasks from past international programming contests. According to the circumstances of the course, problem 1 was not set in weeks 3 and 6.

Table 1: Number of students in group A/B who submitted correct code by problem.

Problem	A	B	Problem	A	B
1-1	14	11	4-1	10	12
1-2	9	8	4-2	9	13
1-3	4	4	4-3	1	5
2-1	11	11	5-1	12	12
2-2	8	11	5-2	9	11
2-3	2	1	5-3	0	0
3-2	10	11	6-2	6	8
3-3	1	2	6-3	0	0

Note: A row (*W-P*) means the problem (*P*) in the week (*W*). Gray color shows the results for problem 3 of each week, which we excluded as noise.

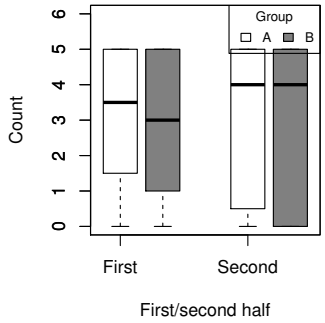


Figure 3: Number of problems solved by group in the first half (1-1 to 3-2) and the second half (4-1 to 6-2).

During the experiments, students were not provided with a reward such as prizes or bonus grades. In other words, participation in the experiment and use of the WOJS was voluntary. This point was emphasized in the first lecture. Note that submissions of answers to the problems, or assignments, were included in the teacher's grading of students, which is standard for the course.

After the experimental period, we collected all submitted code that passed the tests given by the online judge. Failed code was excluded from the data because its CC could not be compared with the CC of the correct code. We then analyzed the CC scores.

4.2 Results

Table 1 shows the number of students in each group who submitted correct code by problem. The problem column indicates the type of the problems in a given week. For example, 1-2 means the 2nd problem in the 1st week. The number of students who submitted correct code depended on the problem. Few students submitted correct answers to the 3rd group of problems because they were too difficult for the students to solve. Since the number of submitted answers for problem 3 each week was too small to analyze statistically, we excluded the results of problem 3.

Figure 3 summarizes the number of problems solved by the students in each half as a box plot. Since the number of problems in each half (except the one excluded as noted above) is 5, the plot falls ranges from 0 to 5. Group A and group B have similar submissions.

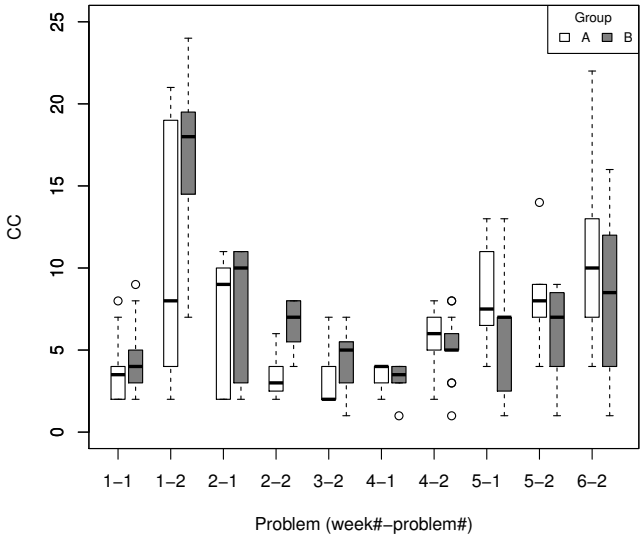
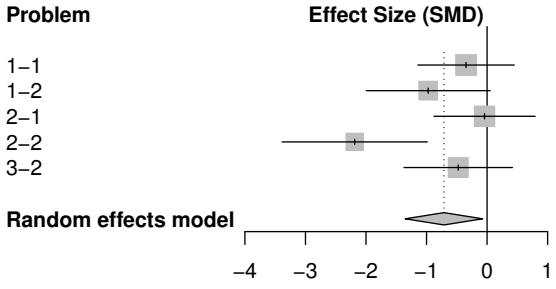
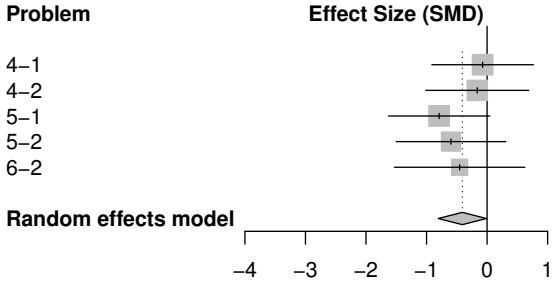


Figure 4: Summary of the CC score by group for each problem.



(a) First half of the course.
(Combined SMD is −0.71. Combined 95% CI is [−1.35, −0.07].)



(b) Second half of the course.
(Combined SMD is −0.41. Combined 95% CI is [−0.81, −0.01].)

Figure 5: SMD of the CC score combined by meta-analysis.
Note: SMD = 0: no effect, SMD < 0: intervention lowered CC.

The p-value by the Wilcoxon-Mann-Whitney test is 0.5985 in first half and 0.7400 in second half. Therefore, we cannot find significant difference in the number of solved problems.

Figure 4 presents the CC scores associated with each problem as a box plot. During the first half of the course, the CC of group A tended to be smaller than that of group B. After changing the intervened group, the CC of group B became small. However, the CC varied depending on the problem. We could not compare the CC scores directly because the problems and the answers differed. Thus, we applied a statistical method from meta-analysis[1] and combined the results.

Figure 5 shows the forest plot for each half of the course where a line represents the outcome of each problem. The outcome was represented as the standardized mean difference (SMD), which is called the effect size of each problem. That is, each effect size shows the size of the CC of the intervention group compared to that of the control group. An effect size less than 0 means that the intervention lowered the CC of the group. Each plot shows the effect size, and the horizontal line segment shows the 95% confidence interval (CI) of the effect size. The size of a square mark indicates the weight for combined effect size. The rhombus shape at the bottom shows the 95% CI of the combined effect size. It takes into account both the within-problem variance and the inter-problem variance.

Figure 5a shows the analysis of the first half of the course. Only the students in group A used WOJS in the first half. The CC of code written by group A was smaller than that of group B. The combined 95% CI was also less than 0. In the second half of the course (Fig. 5b), only the students in group B used WOJS. The CC of code written by group B was significantly lower than that of group A during the second half of the course. Therefore, gamification intervention lowered the CC of code written by the students.

We compared typical a student in each group. The student from group A (Student A6) wrote a low-CC code under the intervention (Fig. 6a), but not after the intervention (Fig. 6b). In contrast, the typical student from group B (Student B4) did not written a low-CC code before the intervention (Fig. 6c) but did during the intervention (Fig. 6d). Both students reduced the CC by using arrays and functions only in the intervention period.

We also analyzed the code submitted by other students. We read the code and compared small-CC code with large-CC code produced for each problem. The students used several techniques to reduce the CC of their code:

- Reducing redundant `if` statements.
- Reducing redundant conditional expressions.
- Rewriting multiple `if` statements in fewer `if` statements with `for` statements.
- Creating new functions to remove duplicate code.
- Using an array of function pointers to branch.

We did not anticipate the function pointers method. One student in group B used this technique to lower the CC score to 1. This

[1] Meta-analysis is a method to combine the results of multiple studies on the same subject [10, 12]. After collecting results, meta-analysis calculates the effect size of each study. When the outcome is continuous data, the standardized mean difference (SMD) can be set as the effect size. SMD is calculated using the following formula:

$$d = \frac{\overline{X}_1 - \overline{X}_2}{\sigma_{\text{pooled}}}$$

where $\overline{X}_1 - \overline{X}_2$ is the difference in means and σ_{pooled} is the pooled standard deviation (SD). Then meta-analysis combines the effect sizes by a weighted mean. The multiplicative inverse of the variance in each study is used as the weight. We used a forest plot [16] to summarize the results of this meta-analysis.

```c
void A(int x) {
  if (x == -1) exit(0);
}

int main(void) {
  int x;
  char a[8][3] = {"N", "NW", "W", "SW", "S", "SE", "
      E", "NE"};
  while (1) {
    scanf("%d", &x);
    A(x);
    x = x / 45;
    printf("%s\n", a[x]);
  }
  return 0;
}
```

(a) 2-1 by A6 (under the intervention).

```c
int main(void) {
  int n, i, A, E;
  scanf("%d", &n);
  for (i = 0; i < n; i++) {
    scanf("%d %d", &A, &E);
    if (A + E == 0) {
      printf("G\n");
      continue;
    }
    switch ((A + E) / 10) {
      case 10:
        printf("A+\n");
        break;
      case 9:
        printf("A+\n");
        break;
      case 8:
        printf("A\n");
        break;
      case 7:
        printf("B\n");
        break;
      case 6:
        printf("C\n");
        break;
      default:
        printf("F\n");
        break;
    }
  }
  return 0;
}
```

(b) 5-1 by A6.

```c
int main(void) {
  while (1) {
    int d;
    scanf("%d", &d);
    if (d == 0) printf("N\n");
    if (d == 45) printf("NW\n");
    if (d == 90) printf("W\n");
    if (d == 135) printf("SW\n");
    if (d == 180) printf("S\n");
    if (d == 225) printf("SE\n");
    if (d == 270) printf("E\n");
    if (d == 315) printf("NE\n");
    if (d == -1) break;
  }
  return 0;
}
```

(c) 2-1 by B4.

```c
void grade(int point);

int main(void) {
  int N, i;
  int A, E;

  scanf("%d\n", &N);

  for (i = 0; i < N; i++) {
    scanf("%d %d\n", &A, &E);
    grade(A + E);
  }
  return 0;
}

void grade(int point) {
  char grade[12][4] = {"G\n", "F\n", "F\n", "F\n", "
      F\n", "F\n", "F\n", "C\n", "B\n", "A\n", "A+\n
      ", "A+\n"};

  point = (point != 0) * (point + 10);
  point /= 10;
  printf("%s", grade[point]);
}
```

(d) 5-1 by B4 (under the intervention).

Figure 6: Code submitted for problem 2-1/5-1 by student A6/B4.
Note: `#include` directives are removed due to space limitations in the paper.

method can reduce the CC, but it does not improve the quality of the code because it just replaces all `if`, `for`, and `while` statements one by one. For example, the following C program calls the function `f2` only if `x` is `0` without an `if` statement.

```c
void (*func[2])();
func[0] = f1;
func[1] = f2;
func[!x]();
```

5 DISCUSSION

We discuss the results in according with the research questions.

> **RQ1** Does an online judge with a leaderboard using the CC change the number of problems solved by each student?

We compared the number of problems solved by each student in group A and that in group B by the Wilcoxon-Mann-Whitney test. We could not find a statistically significant difference both in the first half and in the second half. Moreover, the difference in the actual values seems very small. Thus, our approach does not interfere with students' grades (i.e., the number of problems solved). However, we cannot confirm that there is no difference statistically.

> **RQ2** Does an online judge with a leaderboard using the CC change the CC of code written by students?

The CC of the code written by group A was lower than that of group B in the first half of the course. The difference was significant at the 5% level. The CC of the code written by group B was smaller than that of group A at a 5% significance level in the second half of the course. Therefore, the leaderboard using the CC can reduce the CC of code written during the intervention. Thus, the score and the leaderboard modify behavior in programming education without any additional rewards.

> **RQ3** Does an online judge with a leaderboard using the CC change the code quality?

After reading the submitted code, we found that students implemented several techniques to improve the CC scores. For example, in the intervention period, some students tried dividing the main function into multiple functions and avoiding duplicated code. The techniques found in the students' code would be the first step in learning code quality improvement.

However, we observed that one student stuck to reduce the CC without improving the code quality. The competitive environment of the leaderboard caused students to lower their CC greedily. Such a behavior may occur even when using other simple code metrics such as LOC. One solution is to choose advanced code metrics or a combination of metrics that students cannot improve simplistically. We plan to find suitable metrics to motivate students to write high-quality code in future.

6 THREATS TO VALIDITY

An internal threat to validity is the novelty effect. The novelty effect is a bias for new technology [8]. Although we confirmed that our approach works well in three weeks, it is possible that the effect of our approach will be reduced due to the novelty effect. We plan to conduct a longer experiment to assess the impact of the novelty effect.

The experiment was conducted in a single programming course, which is a threat to the external validity. We gathered participants from students in a programming course at our university. Moreover, not all the students in the course participated in the experiment because some students did not submit a consent form. However, the proposed method can be applied to other courses. In the future, we will evaluate the effectiveness of gamification elements in other groups of students.

Another threat to external validity is that the selection of the programming problem might have been biased. To minimize the bias among the problems, we used multiple problems in each half of the course and combined the outcomes. However, we could not analyze the results of the third-level problems due to the limited number of submissions. We plan to conduct an experiment with students who can solve more difficult problems.

7 RELATED WORK

Several researchers have mentioned code quality in programming education. Breuker et al. compared the static code quality of first year students to that of second year students [5]. They pointed out that programming courses do not emphasize code quality. Zhou et al. revealed the weak points of a previous online judge such as its inattention to code quality [29]. They designed a framework for a new online judge, with four new features: personalized feedback, code quality check, code similarity check, and teaching adjustment. We also improved an online judge by adding a leaderboard with a code metric measurement. Although Zhou et al. did not implement an online judge and did not conduct an experiment, we conducted an experiment to advance the research of code quality in programming education.

There are several practical publications on gamification in software engineering. Singer and Schneider proposed using gamification in a version control system [26]. They developed a web application using a leaderboard to motivate participating students to make more commits. Arai et al. created a gamification tool to motivate developers to remove warnings of static bug finders [4]. They gave points to developers who made commits that solve the warnings. Prause et al. conducted an experiment to evaluate gamification used to enforce coding conventions [22]. They used a score calculated from the number of violations, and developers improved their scores during an intervention. Pedreira et al. showed the systematic mapping of research on gamification in software engineering [20]. They claimed that most studies lack the methodological support to make their proposals reproducible in other settings.

Gamification has also been studied in programming education. Khaleel et al. stated that gamification encourages students to learn a programming language [14]. They designed a gamification application for programming education. Olsson et al. employed a gamification e-learning tool in programming course [19]. Their web application visualized the progress of learning with progress bars and badges. They pointed out that badges induced positive feeling for the course. Rojas et al. created a multiplayer game that teaches mutation testing [24, 25]. They employed point system, which gave scores to winner of each match.

These gamification studies in software engineering and programming education are related to our work. However, we did not find any study combining gamification with code metrics measurement for educational purposes as far as we investigated. Consequently, our paper should contribute to these research fields.

8 CONCLUSIONS

Students are usually motivated to write and submit correct programs by grades in programming courses. However, when a grade is not suitable to motivate a target behavior, gamification can provide another incentive. Here, we propose employing a leaderboard with code metric measurements to motivate students to improve code quality. We developed a web application, which measures the CC of code submitted to an online judge. We conducted a six-week experiment in an intermediate programming course to evaluate our approach.

We collected the submitted code, which passed the tests executed by the online judge through the experiment. We applied meta-analysis to the CC of submissions for each programming problem. Students wrote code with a lower CC under gamification conditions. Therefore, our approach motivates students to improve code metrics without additional rewards. The results also suggest that a leaderboard using the CC does not interfere with students' grades.

In the future, we will design another gamification experiment. To evaluate the effect of gamification (including the novelty effect) more accurately, long-term experiments with more control groups are necessary. In addition, code metrics to measure code quality precisely should be evaluated.

ACKNOWLEDGMENTS

This work was partially supported by JST Presto Grant Number JPMJPR14D4. I am deeply grateful to Prof. Kana Shimizu of Waseda University for the opportunity to conduct this experiment. Students' participation was also invaluable. Advice given by Prof. Ryo Okada of Kagawa University was of great help in the evaluation process.

REFERENCES

[1] 2003. Google Code Jam. https://code.google.com/codejam/

[2] 2010. Codeforces. http://codeforces.com/

[3] 2012. Lizard code complexity analyzer. http://www.lizard.ws/

[4] Satoshi Arai, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. 2014. A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools. In *2014 6th International Workshop on Empirical Software Engineering in Practice*. IEEE, 37–42. https://doi.org/10.1109/IWESEP.2014.17

[5] Dennis M. Breuker, Jan Derriks, and Jacob Brunekreef. 2011. Measuring static quality of student code. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*. ACM Press, New York, New York, USA, 13. https://doi.org/10.1145/1999747.1999754

[6] E. T. Chen. 1978. Program Complexity and Programmer Productivity. *IEEE Transactions on Software Engineering* SE-4, 3 (may 1978), 187–194. https://doi.org/10.1109/TSE.1978.231497

[7] S.R. Chidamber and C.F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (jun 1994), 476–493. https://doi.org/10.1109/32.295895

[8] Richard E Clark and Brenda M Sugrue. 1988. Research on Instructional Media, 1978-1988. In *Educational Media Yearbook 1987-88*. Littletown, CO: Libraries Unlimited, 327–343. https://www.gwern.net/docs/psychology/1991-clark.pdf

[9] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. 2011. From game design elements to gamefulness: defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*. ACM Press, New York, New York, USA, 9. https://doi.org/10.1145/2181037.2181040

[10] Gene V. Glass. 1976. Primary, Secondary, and Meta-Analysis of Research. *Educational Researcher* 5, 10 (nov 1976), 3–8. https://doi.org/10.3102/0013189X005010003

[11] Maurice H. (Maurice Howard) Halstead and Maurice H. 1977. *Elements of software science*. Elsevier. 127 pages. https://dl.acm.org/citation.cfm?id=540137

[12] Larry V. Hedges and Ingram. Olkin. 1985. *Statistical methods for meta-analysis*. Academic Press. 369 pages. https://www.sciencedirect.com/science/book/9780080570655

[13] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research - Koli Calling '10*. ACM Press, New York, New York, USA, 86–93. https://doi.org/10.1145/1930464.1930480

[14] Firas Layth Khaleel, Noraidah Sahari@ Ashaari, Tengku Siti Meriam, Tengku Wook, and Amirah Ismail. 2015. The study of gamification application architecture for programming language course. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication - IMCOM '15*. ACM Press, New York, New York, USA, 1–5. https://doi.org/10.1145/2701126.2701222

[15] Andy Kurnia, Andrew Lim, and Brenda Cheang. 2001. Online Judge. *Computers & Education* 36, 4 (may 2001), 299–315. https://doi.org/10.1016/S0360-1315(01)00018-5

[16] S Lewis and M Clarke. 2001. Forest plots: trying to see the wood and the trees. *BMJ (Clinical research ed.)* 322, 7300 (jun 2001), 1479–80. http://www.ncbi.nlm.nih.gov/pubmed/11408310http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC1120528

[17] T. J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (dec 1976), 308–320. https://doi.org/10.1109/TSE.1976.233837

[18] Allan Mori, Gustavo Vale, Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo, Elder Cirilo, Pooyan Jamshidi, and Christian Kastner. 2018. Evaluating domain-specific metric thresholds. In *Proceedings of the 2018 International Conference on Technical Debt - TechDebt '18*. ACM Press, New York, New York, USA, 41–50. https://doi.org/10.1145/3194164.3194173

[19] Marie Olsson, Peter Mozelius, and Jonas Collin. 2015. Visualisation and Gamification of e-Learning and Programming Education. *Electronic Journal of e-Learning* 13, 6 (2015), 441–454. https://eric.ed.gov/?id=EJ1087309

[20] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. 2015. Gamification in software engineering âĂŞ A systematic mapping. *Information and Software Technology* 57 (jan 2015), 157–168. https://doi.org/10.1016/J.INFSOF.2014.08.007

[21] Jordi Petit, Omer Giménez, and Salvador Roura. 2012. Jutge.org: an educational programming judge. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*. ACM Press, New York, New York, USA, 445. https://doi.org/10.1145/2157136.2157267

[22] Christian R. Prause and Matthias Jarke. 2015. Gamification for enforcing coding conventions. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, New York, New York, USA, 649–660. https://doi.org/10.1145/2786805.2786806

[23] Miguel A Revilla, Shahriar Manzoor, and Rujia Liu. 2008. Competitive Learning in Informatics: The UVa Online Judge Experience. *Olympiads in Informatics* 2 (2008), 131–148. https://ioinformatics.org/journal/INFOL035.pdf

[24] Jose Miguel Rojas and Gordon Fraser. 2016. Code Defenders: A Mutation Testing Game. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 162–167. https://doi.org/10.1109/ICSTW.2016.43

[25] José Miguel Rojas, Gordon Fraser, Thomas White, and Benjamin Clegg. 2016. Code Defenders. http://code-defenders.org/

[26] Leif Singer and Kurt Schneider. 2012. It was a bit of a race: Gamification of version control. In *2012 Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques (GAS)*. IEEE, 5–8. https://doi.org/10.1109/GAS.2012.6225927

[27] Armando M Toda, Pedro H D Valle, and Seiji Isotani. 2018. The Dark Side of Gamification: An Overview of Negative Effects of Gamification in Education. In *Higher Education for All. From Challenges to Novel Technology-Enhanced Solutions*, Alexandra Ioana Cristea, Ig Ibert Bittencourt, and Fernanda Lima (Eds.). Springer International Publishing, Cham, 143–156.

[28] Sheng Yu and Shijie Zhou. 2010. A survey on metric of software complexity. In *2010 2nd IEEE International Conference on Information Management and Engineering*. IEEE, 352–356. https://doi.org/10.1109/ICIME.2010.5477581

[29] Wenju Zhou, Yigong Pan, Yinghua Zhou, and Guangzhong Sun. 2018. The framework of a new online judge system for programming education. In *Proceedings of ACM Turing Celebration Conference - China on - TURC '18*. ACM Press, New York, New York, USA, 9–14. https://doi.org/10.1145/3210713.3210721