Game-Q: Gamification to Inform Code Quality in an Assessment Submission System

Oscar Karnalim, Rossevine A. Nathasya*, Sendy F. Sujadi, Wenny F. Senjaya, and Erico D. Handoyo

Faculty of Smart Technology and Engineering, Maranatha Christian University, Bandung, Indonesia
Email: oscar.karnalim@it.maranatha.edu (O.K.); rossevine.an@it.maranatha.edu (R.A.N.); sendy.fs@it.maranatha.edu (S.F.S); wenny.fs@it.maranatha.edu (W.F.S.); erico.dh@it.maranatha.edu (E.D.H.)
*Corresponding author

Manuscript received March 20, 2025; revised May 15, 2025; accepted July 8, 2025; published December 19, 2025

***Abstract*—Although code quality is important in industry, it is often overlooked in academia. Several tools have been developed to help students learn the material. However, most of them are not focused on student engagement. We present Game-Q, a gamification tool that integrates code quality assessment into an assessment submission system. After each submission, students will be informed about code quality issues identified in their own work. They can get more game points by submitting high-quality codes or collecting badges (either self-progress or competition). The top ten students with the highest game points will be shown on the leaderboard. Game-Q was evaluated over one year (2023), involving 228 undergraduate students enrolled in Informatics Engineering and Information Systems. They were enrolled in seven programming courses. According to our observations via post-course surveys, we found that Game-Q appears to have a positive effect on students' motivation and their awareness of our standards for code quality.**

***Keywords*—gamification, code quality, programming, engineering education**

I. Introduction

Software should be maintained in the industry, and it can be easier if the code is high quality [1]. However, some early-career software developers may struggle to write high-quality code. One possible reason is that code quality is often overlooked in academia [2]. Some courses encourage students to write high-quality code, but it is less prioritized [3].

Several tools have been developed to promote code quality. For instance, Flake8 [4] and checkstyle [5] are two common code quality checkers. Although they are useful for advanced programmers, the feedback might be less relevant to novice programmers, such as university students. Consequently, a number of dedicated code quality checkers for academia have been developed. Hyperstyle [6] assesses student submissions and their quality in four categories: excellent, good, moderate, and bad. GRADESTYLE [7] expands the checkstyle with several additional checks, including unused code, string concatenation, and code clones. Refactor Tutor [8] educates students about code quality with case studies and gradual hints.

WebTA is a programming IDE with code quality feedback [9]. FrenchPress is an Eclipse plug-in that reports four code quality issues covering field, modifier, loop, and Boolean [10]. CQIS [11] reports code quality issues upon each submission. RTSF [12] incorporates large language models to suggest better identifier names. AutoStyle allows instructors to select the code quality rules based on student historical data [13].

Existing tools are primarily focused on the learning process. However, students need to be engaged with them. In many courses, the use of such tools should be voluntary. Even if tool use can be strongly encouraged, some students might not be engaged with them and will have minimal learning improvement.

In response to the aforementioned gap, we developed a gamification to inform code quality (Game-Q, in short). Three game components are considered: points, badges, and a leaderboard [14]. Students can earn more points and badges by submitting high-quality code, especially if it exceeds the quality of their colleagues. The top ten students with the highest game points will be displayed on a leaderboard. To the best of our knowledge, this is the first instance of gamification being applied to code quality. Some gamification studies are dedicated to programming, but the main focus is on the ability to program [15].

In 2023, Game-Q was employed in seven programming courses, involving 228 undergraduate students. It was evaluated based on a post-survey asking about student motivation and awareness of our code quality standards.

II. Literature Review

A. *Gamification*

Gamification is the integration of game elements into non-game situations [16, 17]. Game elements include **feedback**, **reputation, rankings, levels, time pressure**, and **competition with explicit and enforced rules [16, 18].** They have various engagement levels for the students. Badges, for example, might be less motivating for students over time [19]. Several studies suggest that gamification can be beneficial in education [14, 15], though implementing it can be sometimes challenging [20]. Students tend to complete their assessments faster to earn points or rank higher [21].

Papadakis [22] developed a gamification to introduce programming in secondary education. Students interact with the system and create their relevant content. According to their quasi-experiment, gamification had a positive effect on students' programming skills and their motivation to learn programming.

Castellano *et al.* [23] developed a mobile gamification application to teach human anatomy to students. The application featured a recommendation system and a virtual assistant. They found that students benefited from the gamification. Takacs *et al.* [24] developed gamification to support their online learning, especially soft skills and self-motivation.

Harrington and Chaudhry [25] introduced the use of points in games as part of their teaching methods. To support the implementation of this concept, they developed an application called TrAcademic, featuring a public leaderboard that openly displays students' points and achievements. In this system, several categories of points are applied. **Experience Points** are awarded to students for attendance and completing basic exercises. **Challenge Points** are given to those who complete more difficult challenges. Meanwhile, **Teaching Points** are earned by students who assist their peers in understanding the course material. The results showed increased **weekly student attendance** and improved academic performance [26].

Irwin and Edwards [27] studied using an energy bar system in programming assignments inspired by mechanics commonly found in mobile games. Students could only submit assignments if they had enough energy. Each submission consumed 1 unit of energy, with a maximum limit of 3 units at a time. Energy is replenished at a rate of 1 unit per hour. If a student ran out of energy but was approaching the deadline, they were still allowed to submit their assignment within the final hour before the deadline. The results showed a change in students' work patterns, encouraging them to start assignments earlier, work more consistently, and reduce procrastination habits.

Chans and Castro [28] incentivised student engagement in an online learning platform for chemistry courses. Students were more engaged and thus had better performance in the courses. Cigdem *et al.* [29] introduced a leaderboard as part of their gamification for engineering students in completing formative assessments. They were more engaged with the assessments, though it might not keep their attention for long.

De Pontes *et al.* [26] conducted a study to determine whether gamification elements (leaderboards, badges, and personal record tracking) influence student engagement in a course. The results showed that the gamified group completed more tasks than the non-gamified group.

Several other studies have also proven that gamification can enhance student learning outcomes by encouraging higher engagement in completing assignments and spending more time studying [30–33].

Although gamification has its benefits, it can also lead to several negative impacts [34], including a gradual decline in student motivation [35], indifference [30, 36, 37], the emergence of undesired behavior [38], loss of performance [39, 40]. These impacts arise due to the lack of a well-developed gamification design.

Self-Determination Theory (SDT) is one of the most frequently applied motivation theories in gamification for education [41, 42]. This theory highlights three primary psychological needs that support motivation in gamification: autonomy, competence, and relatedness. Autonomy refers to the need for students to feel a sense of control over their decisions and actions within the gamification process. Competence is the feeling of being capable of achieving success, where students understand that the goals within gamification are attainable through reasonable effort. Relatedness emphasizes the importance of social connections, allowing students to feel like part of a community within the gamification system. SDT also explains that goal-oriented behavior is influenced by two main types of motivation: intrinsic and extrinsic [43]. Intrinsic motivation is the natural drive to engage in an activity due to personal interest, while extrinsic motivation arises from external incentives or rewards. Interestingly, extrinsic motivation can evolve into intrinsic motivation when an individual initially participates in an activity for external rewards but gradually begins to enjoy it and continues due to genuine interest.

B. Code Quality Checkers

There are several tools designed to assess code quality, including: Hyperstyle [6], GRADESTYLE [7] and WebTA [9]. The evaluation of Hyperstyle [6] was conducted in two main scenarios. First, the tool was compared with the Tutor [8]. The second scenario focused on evaluating the impact of Hyperstyle on students' code quality. The results showed that Hyperstyle significantly contributed to improving students' code quality. It detected more formatting-related errors and provided more suggestions for enhancing code quality compared to the Tutor application.

GRADESTYLE is a tool that automatically checks and evaluates code style in programming assignments [7]. It utilizes checkstyle [5] and PMD [44] to detect specific categories of violations, JAVAPARSER [45] to identify variable constructions, and EXTJWNL [46] to detect issues in the class and method names. An evaluation was conducted using GRADESTYLE in a second-year software engineering programming class with 327 students. The results showed a consistent improvement in code style across student assignments, as well as an overall enhancement in code quality.

WebTA is designed to detect antipatterns and provide suggestions for improvement [9]. It implements several antipatterns, including Misplaced Code, Pseudo-Implementation, Localized Instance Variable, Knee-Jerk, and Repeated Resource Instantiation. Misplaced Code refers to code placed in an inappropriate location within the program structure. At the same time, Pseudo-Implementation describes code that appears to implement a feature but does not function correctly or merely acts as a placeholder. Localized Instance Variable occurs when instance variables are used in a scope where local variables would suffice, leading to unnecessary memory consumption. Knee-Jerk involves excessive use of conditions or statements without an apparent reason, such as redundant checks that are not needed. Meanwhile, repeated resource instantiation occurs when the same object or resource is repeatedly created within a program, resulting in inefficiency and excessive memory usage. By identifying these antipatterns, WebTA provides meaningful feedback, enabling beginner programmers to recognize and correct their mistakes more effectively, ultimately improving their coding practices.

III. Methods

Game-Q is integrated into a web-based assessment submission system that accepts Java/Python programs [11]. It generates code quality reports for each submission. Instructors can create and manage courses, assignments, and student submissions. They can also enroll students on courses. Students will get a code quality report sent to their email address upon submission. For convenience, the report is in an HTML format and can be opened without a login. Students can learn from the identified code quality issues and thus avoid repeating the same mistakes in future assignments.

They can also resubmit the programs if they want to. The system is developed in PHP, HTML, JavaScript, and CSS. The database is stored in MySQL as a set of relational tables.

Game-Q is inspired by Karnalim *et al*. [47] and it covers 32 Java and 20 Python code quality issues. The Java quality issues are detected using checkstyle, while the Python quality issues are detected using flake8. All identified issues are dedicated to novice programmers, covering comments, white space, identifiers, expressions, branching, looping, and shorthand syntax. Two examples are empty syntax blocks and overly deep nested loops. The former is not recommended, as each syntax block should have contents. The latter is not recommended, as it is difficult to follow the program's flow, especially when logical errors occur. We acknowledge that there is no consensus about code quality. Hence, we define our own standards based on consensus among instructors.

Students will get game points by completing assignments and/or obtaining badges. A code quality score for each submission will be calculated as in Eq. (1) and added to the student's game points. The score will be between 0 and 100 inclusive. If no issues are identified, the score will be 100, the highest possible. However, any identified issues will reduce the score by the proportion of a syntax block to the program size. The number of tokens for each syntax block is modifiable per programming language. However, it is set to cover eight program statements by default: 80 tokens for Java, as one Java statement typically spans around 10 tokens; 40 tokens for Python, as one Python statement typically spans around 5 tokens. These numbers are taken from another study [11]. As an illustration, if a student program has 5 issues and its program size is 800 tokens, the code quality score will be $100-(5\times100\times80/800)=50$.

$$CQ = max(0,100-(|issue|\times100\times|block|/|prog|)) \qquad (1)$$

The code quality scores for the same assignment will be averaged to prevent students from manipulating game points by submitting multiple times within a single assignment. Students can also get more game points by obtaining badges. Each badge is worth 100 game points.

There are two badge categories: self-progress and competition. They can only be obtained once per assignment. Self-progress badges reflect students' own progress in learning code quality. A "High-quality submission" badge is given to students with code quality scores above 75. A "Quality wisdom" badge is given to those who accessed the code quality report. Competition badges are awarded to the three students with the highest code quality scores for a given assignment. They will be provided after the assignment closes. The first will be awarded "Finest quality submission"; the second will be awarded "Second-finest quality submission"; and the third will be awarded "Third-finest quality submission". Those who submitted the programs earlier will be preferred if several students have equally high code quality scores.

Fig. 1 shows the game badges. The leftmost badge is for "high-quality submission", followed by those for "quality wisdom", "Finest quality submission", "Second-finest quality submission" and "Third-finest quality submission".

Each student will be assigned a game level, calculated as in Eq. (2). It is simply an average of the obtained game points with 300. The justification is that, for each assignment, students can earn 100 game points by submitting high-quality submissions and an additional 200 points from self-progress badges. There is no maximum level limit for the game.

$$Lvl = game\_point/300 \qquad (2)$$

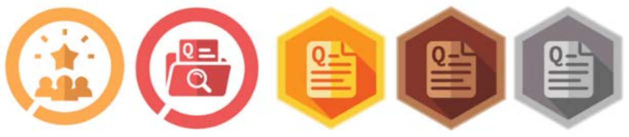
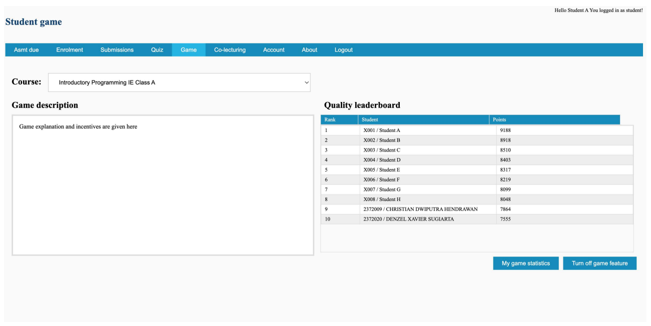


Fig. 1. Game badges.

Students can see their total game points and badges. They can also see the top ten students with the highest game points in the leaderboard, sorted in descending order. This is to encourage students to obtain more game points. We encouraged each course to incentivize students on the top list.

Fig. 2 shows the gamification page. On the left side, there will be a general explanation of the gamification and the incentives. On the right side, the leaderboard displays ten students with the highest game points, sorted in descending order. Students enrolled in multiple courses can check their progress in each course by clicking the drop-down at the top of the page. It will show all enrolled courses. At the bottom, students can view their game statistics in detail or disable the game feature.



Course: Introductory Programming IE Class A

Game description

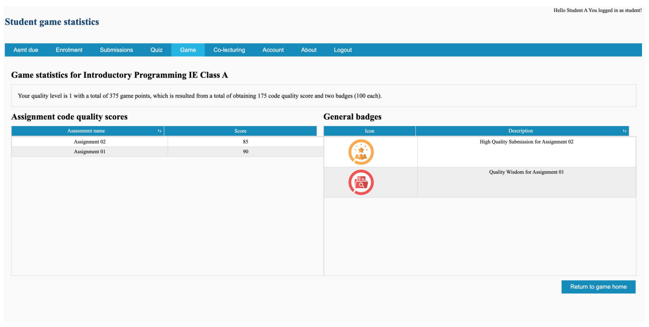
Game explanation and incentives are given here

Quality leaderboard

Rank	Student	Points
1	X001 / Student A	9188
2	X002 / Student B	8918
3	X003 / Student C	8510
4	X004 / Student D	8403
5	X005 / Student E	8317
6	X006 / Student F	8219
7	X007 / Student G	8099
8	X008 / Student H	8048
9	2372009 / CHRISTIAN DWIPUTRA HENDRAWAN	7864
10	2372020 / DENZEL XAVIER SUGIARTA	7555

Fig. 2. Gamification page.

The game statistics page is shown in Fig. 3, displaying the game progress of a particular student. At the top of the page, students will be informed about the number of game points they have obtained, including the breakdown by code quality scores and badges. The two tables at the bottom show the details. The code quality score for each assignment will be displayed on the left side of the page. All obtained badges will be listed along with their assignment information on the right side.



Game statistics for Introductory Programming IE Class A

Your quality level is 1 with a total of 375 game points, which is resulted from a total of obtaining 175 code quality score and two badges (100 each).

Assignment code quality scores

Assessment name	Score
Assignment 02	85
Assignment 01	90

General badges

Icon	Description
	High Quality Submission for Assignment 02
	Quality Wisdom for Assignment 01

Fig. 3. Game statistics page.

Students can turn off the game feature if they are not interested in participating in the gamification. The button acts like a toggle. If a student does not participate in the gamification, they will not be able to see their game progress or the game leaderboard. Further, they will not be counted in the leaderboard. However, their game points will still be calculated on the server, so they can join the gamification later without losing any game points.

Game-Q implements SDT in its mechanics. Students' autonomy is reflected in their decisions to participate in the gamification. They can also choose to submit their work to earn more game points. Competence is reflected in the leaderboard and the game points. Students can easily earn some game points, but require extra effort to stay on the leaderboard. Relatedness is reflected in the leaderboard and the badges. Students can observe their colleagues' performances and may ask them for guidance.

While our gamification is primarily designed for novice programmers, we believe it can also be beneficial for experienced programmers. They can be reminded if they accidentally missed some code quality considerations. To promote engagement among experienced programmers, the game's rankings, points, and badges could be enabled for social media sharing (e.g., LinkedIn or GitHub).

IV. Evaluation

We were interested in evaluating the impact of Game-Q on student motivation and awareness of our code quality standards. Hence, it was employed in seven programming courses at the institution of the first author in 2023; details are provided in Table 1. They were from two majors: Informatics Engineering (IE) and Information Systems (IS). In all courses, an assignment would be given for each session, consisting of up to three programming tasks. One assignment would typically be completed in under two hours. At the end of each course, the top five students with the highest game points would be awarded a 5-point bonus for the overall assignment mark.

Table 1. Courses employing Game-Q

ID	Course	Students	Year of study
IP (IE)	Introductory Programming (IP) IE	48	First-year
IP (IS)	Introductory Programming (IP) IS	42	First-year
OOP	Object Oriented Programming (OOP) IS	32	First-year
DS	Data Structure (DS) IE	45	Second-year
MI	Machine Intelligence (MI) IE	34	Third-year
BAP	Business Application Programming (BAP) IS	12	Third-year
SDP	Software Design Pattern (SDP) IE	15	Third-year or fourth-year

At the end of each semester, a post-survey about student motivation and awareness of code quality was distributed. Game-Q was expected to motivate students to write high-quality code and enhance their understanding.

The motivation part of the post-survey consisted of six questions, as shown in Table 2. The first five were Likert-scale questions where 1 refers to "strongly disagree", and 5 refers to "strongly agree". The last question should be responded by either "point", "badge", or "leaderboard". This part primarily addressed students' autonomy in interacting with the system.

The awareness part consisted of 16 questions, as shown in Table 3. The first 15 were Likert-scale questions, while the last question should be responded to by "every time", "while writing the code", "after the code works", and "never". These questions assessed students' competence and relatedness.

Table 2. Motivation survey question

ID	Question
M01	I am motivated to write high-quality code due to the gamification
M02	I am motivated to open a code quality report due to the gamification
M03	I am motivated to complete assignments due to the gamification
M04	I am motivated to collect more game points in the gamification
M05	I am motivated to get high ranks on the leaderboard
M06	Which game element motivates you the most?

Table 3. Awareness survey question

ID	Question
A01	Comments should be clear and easy to understand.
A02	Comments should exist for each program block.
A03	Names of identifiers should be clear and easy to understand.
A04	Names of identifiers should have a consistent transition style.
A05	Each line of code should not be too long.
A06	Each line of code should have only one program statement.
A07	Modules should be imported when used.
A08	Variables declared and assigned with values should be limited to those actually used.
A09	One-line branching should be avoided.
A10	Boolean operators in an expression or a condition for branching/looping should not be too many.
A11	Syntax blocks should not have empty components.
A12	A function body should not have too much code.
A13	Nested branching, looping, or try-catch should not be too deep.
A14	Braces, brackets, and semicolons should be adequately used.
A15	I consider code quality while completing my assignments.
A16	When do you consider code quality while completing your assignments?

Our study's protocols adhered to the ethical guidelines of our institution. Students were informed about the study and were required to provide their consent before participating. The participation was voluntary, and the data were anonymized for privacy. Analysis addressing individuals or smaller groups was not permitted.

In general, Fig. 4 depicts that students were motivated by gamification, though they had the autonomy not to engage with it, except for the Object Oriented Programming (OOP) students. The average Likert score for all courses was 3.94 (very close to 'agree'), while on most courses, the average score was higher than 3.5 (tends to 'agree'). Gamification seemed to be quite helpful in motivating students. OOP students were not very motivated since they had limited spare attention for the gamification. They were still first-year students and needed to complete two weekly OOP assignments. Unlike IP assessments, OOP assessments were expected to have longer and more complex solutions.

On the contrary, the gamification motivated SDP students the most. The students were already aware of the importance of writing high-quality code, which affected their autonomy and thus led them to choose to engage with the gamification. They were either third-year or fourth-year students, and they had some experience in writing large programs.

When observed per the motivation question (see Fig. 5), the gamification motivated students the most to open code quality reports (M02 with an average score of 4.12, 'agree').

From the report, students could understand which quality aspects they had missed. SDP students were still the most motivated, while OOP students were the least motivated.

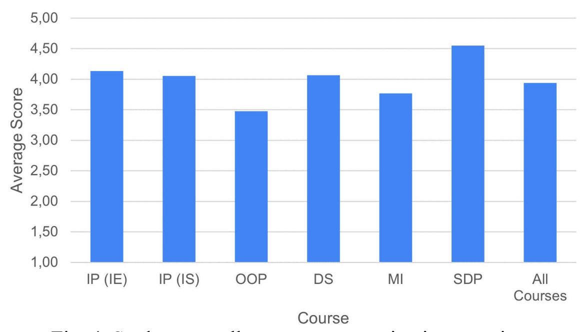


Fig. 4. Student overall responses to motivation questions.

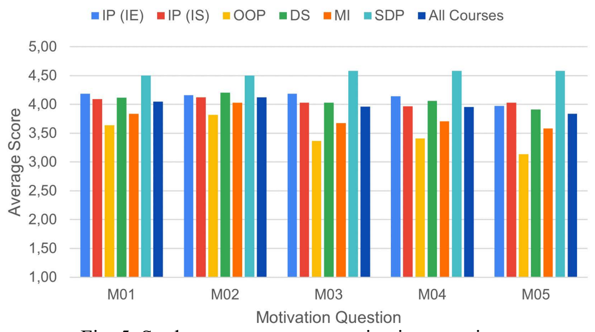


Fig. 5. Student responses per motivation questions.

We also found that students were generally motivated to write high-quality code (M01 with an average score of 4.05, 'agree'). Upon reviewing the code quality reports, students would gain a better understanding of code quality and would thus apply this knowledge in their later submissions. SDP students were still the most motivated.

Students were also motivated to complete assessments (M03) and to obtain more game points than their colleagues (M04). Their average scores were very close to 'agree' (3.96 for M03 and 3.95 for M04). Students could only participate in the gamification by completing assessments. Their interest in obtaining more game points demonstrated that our gamification was effective.

Students were motivated to achieve the highest rank on the leaderboard to demonstrate their competence and sense of belonging to others. However, the motivation was not as high as that for other activities. M05's average score was 3.84. The leaderboard displayed the highest game points, and staying in the first rank became increasingly difficult on later assessments. Students who had obtained many game points at the beginning would likely keep their high rank at the end of the semester. Further study might include bonus points to turn the ranks around.

According to M06 responses (see Fig. 6), the most motivating game element was the leaderboard. This was expected since the leaderboard summarised the students' game performance in a course. Students could observe their performance (competence) and compare it to that of ten students with the highest game points (relatedness). However, the students' purpose of such observation was not to strive for the first rank since being the first-ranked student might be challenging (as confirmed in M05). If students were in the top ten, they would see who had better ranks than them and try to surpass them. Otherwise, they would check the 10th-ranked student and see whether their game points could be surpassed.

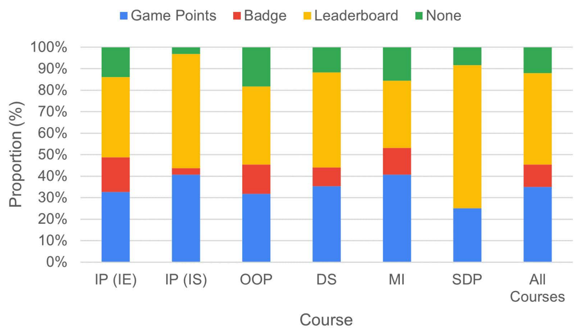


Fig. 6. Student responses for M06.

The leaderboard showed the top ten students based on their game points, so game points were expected to be the second most motivating element, with the main focus on students' competence. Game points were obtained from submitting readable code and/or obtaining badges. They helped the students measure and quantify their performance in gamification. They could also learn how well their particular actions aligned with the goal of the gamification. Some students occasionally checked the game points after completing an assessment or obtaining badges.

Badges were the least motivating game element, which aligned with another study [19]. Students' intrinsic motivation for the badges might decrease over time. Other potential reasons include unclear requirements to obtain particular badges. Students were not provided with detailed instructions on obtaining specific badges, as we wanted them to discuss the matter among colleagues. This might somehow inhibit students' competence and relatedness. Moreover, obtaining badges would affect the game points and the leaderboard. Each badge was worth 100 game points.

Nearly one-eighth of the students were not motivated by the game elements. They might not have been interested in gamification because learning the mechanics was too much of a hassle, or they were not competitive. They had the autonomy to do that. Few of them were not interested in not only the gamification but also engaging with the course. This was quite common in academia. Students enrolled in a course just because it was mandatory or because their families or colleagues asked them. They showed limited interest in the course and its relevant components. Future work might include mechanisms to attract these students to gamification.

Fig. 7 shows that students generally had a good understanding (competence) of our standards for code quality. The average score was 3.88 out of 5. The gamification was somewhat useful for motivating students to write high-quality code and helping them understand the standards despite the fact that they had autonomy not to do so. SDP students had the best understanding of our standards (4.32 out of 5) as they were the most motivated. They were engaged with our gamification and thus obtained the most benefits. Further, they understood the importance of writing high-quality code. They were senior students who had experience in writing large programs.

Although MI students were among the less motivated groups, they had the second-best understanding of our code quality standards (3.94). Like SDP students, they were senior students with experience in software projects. They might not be exceptionally motivated, but they saw the importance of learning such matters (affecting their competence). Students from the remaining courses had a bit lower awareness of code quality, as they were either first-year or second-year students. OOP students were the least aware of code quality, as their motivation was also the lowest (3.62). The gamification needed to be updated to engage with these students.

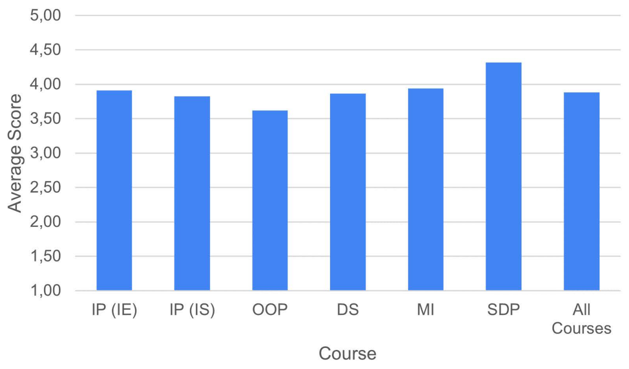


Fig. 7. Student overall responses for awareness questions.

In general, most students understood the need for clear identifier names (A03), clear comments (A01), declaring only needed variables (A08), and reasonable length per code line (A05). They all resulted in average scores of more than 4 ('agree'), showing high competence. Identifiers, comments, and variables were quite common in student code and thus reported by our gamification. While completing assessments near the submission close time, some students tended to write long code lines. Their goal was to complete assessments, not to write high-quality code. Our gamification also reported this, and students became aware not to do that again if there was sufficient time.

When observed per course, SDP students were the most aware of these standards, while OOP students were the least aware. This was consistent with our findings on the overall understanding of code quality standards. Students' competence might be affected by their autonomy to engage with the gamification. Students tended to agree with most of the remaining code quality standards: consistent variable transition in naming (A04), consideration of code quality (A15), importing only used modules (A07), no empty syntax blocks (A11), comment existence (A02), one statement per line (A06), efficient use of brackets and semicolons (A14), limited number of expressions in conditions (A10); no shorthand branching (A09), no overly-deep nested branching or looping (A13). Although their average scores were less than 4 ('agree'), they were higher than 3.5. Our gamification promoted student awareness regarding such standards by reporting code quality issues. On most standards, SDP students were the most aware, while OOP students were the least aware.

A12 (limiting function/method size) was the only standard with an average Likert score lower than 3.5. Most students did not consider this an issue since our gamification seldom reported it. The courses employed a "many small programs" approach. They offered many small assessments instead of large ones. Small assessments were unlikely to have large functions/methods.

Reflecting on SDT, instructors noted that some students asked in detail about how the gamification worked before agreeing to participate. Most participating students consistently submitted their work, demonstrating that they effectively utilised their autonomy. Students also felt that they were competent in achieving higher game points. Some students were very competitive. They checked the leaderboard page after each submission. Students' relatedness was reflected in occasional discussions, especially in how they could get more game points.

We did not record the number of reported code quality issues each week. However, our instructors noted that the number was slightly reduced by about 50%. Students usually had five to ten code quality issues per assessment at the beginning of the semester. Fewer students had more than seven code quality issues in the middle of the semester (around the mid-test). Nearing the end of the semester, students typically only had five code quality issues.

V. Limitations

Our study has several limitations. First, the study did not compare students' performance before and after the integration of Game-Q. Such a comparison could strengthen the findings. Second, while the self-reporting survey was quite valuable for observing the impact of Game-Q, we acknowledged that triangulating the findings with other performance data (such as the number of reported code quality issues each week) might be beneficial. Third, the study is conducted on courses that issue many small programming assignment tasks. Reconducting it on different types of assignments might enrich the findings. Fourth, our code quality standards are specifically designed for novice programmers. Altering the focus to experienced programmers might produce different findings. Fifth, our study was conducted at a particular institution. Reconducting the study at other institutions with different cultural backgrounds in a different country might strengthen our findings. Sixth, while our gamification employs SDT, its impact was not thoroughly evaluated. Future considerations might help complement our findings. Seventh, as gamification is used for the entire semester, some students may become disengaged with it after several weeks. The impact can be improved by maintaining engagement with more game mechanics.

VI. Conclusion and Future Work

We present Game-Q, a gamification to inform code quality. Our experiment involving 228 students found that Game-Q appears to improve students' motivation (autonomy) to write high-quality code. It might also help increase student awareness of our code quality standards (competence and relatedness).

For future work, our study has several directions. First, we plan to focus on unmotivated students and explore the use of bonus points to improve their performance. Second, we aim to promote student interest in badges by describing the mechanics of obtaining them, displaying the obtained badges on students' public profiles, and incorporating more interactive animations. Third, we aim to employ a quasi-experiment that incorporates performance and analytics data to triangulate self-reporting measurements. The data can include the number of submissions, observe quality issues, and collect badges. If possible, the impact of SDT, game elements, and user typology can be properly analysed. Fourth, we want to reconduct the experiment on different courses and students to enrich our findings. Fifth, we are interested in evaluating the technical performance of Game-Q, which includes its efficiency and scalability. Sixth, Game-Q can be redesigned as a module so that it can be easily integrated into other LMS.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Oscar Karnalim and Rossevine A. Nathasya led the research and wrote the paper in most sections. They also polished sections written by other authors. Sendy F. Sujadi, Wenny F. Senjaya, and Erico D. Handoyo conducted experiments in their classes, observed the results, and wrote initial drafts of the findings. All authors had approved the final version.

References

[1] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *Proc. 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 286–28610.

[2] D. Kirk, T. Crow, A. Luxton-Reilly, and E. Tempero, "On assuring learning about code quality," in *Proc. the Twenty-Second Australasian Computing Education Conference*, 2020, pp. 86–94.

[3] H. Keuning, J. Jeuring, and B. Heeren, "A systematic mapping study of code quality in education," in *Proc. the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 2023, pp. 5–11.

[4] Python Code Quality Authority (PyCQA). (Mar. 14, 2025). Flake8: your tool for style guide enforcement. [Online]. Available: https://flake8.pycqa.org/en/latest/

[5] Checkstyle Project. (Mar. 14, 2025). Checkstyle. [Online]. Available: https://checkstyle.sourceforge.io/

[6] A. Birillo, I. Vlasov, A. Burylov *et al.*, "Hyperstyle: A tool for assessing the code quality of solutions to programming assignments," in *Proc. the 53rd ACM Technical Symposium on Computer Science Education*, 2022, pp. 307–313.

[7] C. Iddon, N. Giacaman, and V. Terragni, "GRADESTYLE: GitHub-integrated and automated assessment of Java code style," in *Proc. 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, IEEE, 2023, pp. 192–197.

[8] H. Keuning, B. Heeren, and J. Jeuring, "A tutoring system to learn code refactoring," in *Proc. the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 562–568.

[9] L. C. Ureel II and C. Wallace, "Automated critique of early programming antipatterns," in *Proc. the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 738–744.

[10] H. Blau and J. E. B. Moss, "FrenchPress gives students automated feedback on Java program flaws," in *Proc. the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 2015, pp. 15–20.

[11] O. Karnalim, Simon, W. Chivers, and B. S. Panca, "Automated reporting of code quality issues in student submissions," in *Proc. IFIP World Conference on Computers in Education*, 2023, pp. 517–529.

[12] J. Woodrow, A. Malik, and C. Piech, "AI teaches the art of elegant coding: timely, fair, and helpful style feedback in a global course," in *Proc. the 55th ACM Technical Symposium on Computer Science Education,* 2024, pp. 1442–1448.

[13] R. Roy Choudhury, H. Yin, and A. Fox, "Scale-driven automatic hint generation for coding style," *Intelligent Tutoring Systems (ITS 2016)*, vol. 9684, pp. 122–132, 2016.

[14] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?—A literature review of empirical studies on gamification," in *Proc. 2014 47th Hawaii International Conference on System Sciences, IEEE*, 2014, pp. 3025–3034.

[15] Z. Zhan, L. He, Y. Tong, X. Liang, S. Guo, and X. Lan, "The effectiveness of gamification in programming education: Evidence from a meta-analysis," *Computers and Education: Artificial Intelligence*, vol. 3, 100096, 2022.

[16] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness," in *Proc. the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, 2011, pp. 9–15.

[17] M. Kalogiannakis, S. Papadakis, and A.-I. Zourmpakis, "Gamification in science education. a systematic review of the literature," *Educ Sci (Basel)*, vol. 11, no. 1, p. 22, Jan. 2021.

[18] B. Reeves and J. L. Read, *Total Engagement: How Games and Virtual Worlds Are Changing the Way People Work and Businesses Compete*, Harvard Business Press, 2009.

[19] E. Kyewski and N. C. Krämer, "To gamify or not to gamify? An experimental field study of the influence of badges on motivation, activity, and performance in an online learning course," *Comput Educ*, vol. 118, pp. 25–37, Mar. 2018.

[20] S. Papadakis, "The use of computer games in classroom environment," *International Journal of Teaching and Case Studies*, vol. 9, no. 1, p. 1, 2018.

[21] S. S. Borges, V. H. S. Durelli, H. M. Reis, and S. Isotani, "A systematic mapping on gamification applied to education," in *Proc. the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 216–222.

[22] S. Papadakis, "Evaluating a game-development approach to teach introductory programming concepts in secondary education," *International Journal of Technology Enhanced Learning*, vol. 12, no. 2, p. 127, 2020.

[23] M. S. Castellano, I. Contreras-McKay, A. Neyem *et al.*, "Empowering human anatomy education through gamification and artificial intelligence: An innovative approach to knowledge appropriation," *Clinical Anatomy*, vol. 37, no. 1, pp. 12–24, 2024.

[24] J. Módné Takács, M. Pogátsnik, and T. Kersánszki, "Improving soft skills and motivation with gamification in engineering education," in *Proc. International Conference on Interactive Collaborative Learning*, 2022, pp. 823–834.

[25] B. Harrington and A. Chaudhry, "TrAcademic: improving participation and engagement in CS1/CS2 with gamified practicals," in *Proc. the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 2017, pp. 347–352.

[26] R. G. Pontes, D. D. S. Guerrero, and J. C. A. Figueiredo, "Analyzing gamification impact on a mastery learning introductory programming course," in *Proc. the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 400–406.

[27] M. S. Irwin and S. H. Edwards, "Can mobile gaming psychology be used to improve time management on programming assignments?" in *Proc. the ACM Conference on Global Computing Education*, 2019, pp. 208–214.

[28] G. M. Chans and M. P. Castro, "Gamification as a strategy to increase motivation and engagement in higher education chemistry students," *Computers*, vol. 10, no. 10, p. 132, 2021.

[29] H. Cigdem, M. Ozturk, Y. Karabacak, N. Atik, S. Gürkan, and M. H. Aldemir, "Unlocking student engagement and achievement: the impact of leaderboard gamification in online formative assessment for engineering education," *Educ Inf Technol (Dordr)*, vol. 29, no. 18, pp. 24835–24860, 2024.

[30] L. de-Marcos, A. Domínguez, J. Saenz-de-Navarrete, and C. Pagés, "An empirical study comparing gamification and social networking on e-learning," *Comput Educ*, vol. 75, pp. 82–91, 2014.

[31] L. Rodrigues, A. M. Toda, W. Oliveira, P. T. Palomino, A. P. Avila-Santos, and S. Isotani, "Gamification works, but how and to whom? An experimental study in the context of programming lessons," in *Proc. the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 184–190.

[32] P. Denny, F. McDonald, R. Empson, P. Kelly, and A. Petersen, "Empirical support for a causal relationship between gamification and learning outcomes," in *Proc. the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–13.

[33] R. N. Landers and A. K. Landers, "An empirical test of the theory of gamified learning: the effect of leaderboards on time-on-task and academic performance," *Simul Gaming*, vol. 45, no. 6, pp. 769–785, 2014.

[34] A. M. Toda, P. H. D. Valle, and S. Isotani, "The dark side of gamification: An overview of negative effects of gamification in education," *FAPESP*, 2018, pp. 143–156.

[35] A. Domínguez, J. Saenz-de-Navarrete, L. de-Marcos *et al.*, "Gamifying learning experiences: practical implications and outcomes," *Comput Educ*, vol. 63, pp. 380–392, Apr. 2013.

[36] L. Haaranen, P. Ihantola *et al.*, "How (not) to introduce badges to online exercises," in *Proc. the 45th ACM Technical Symposium on Computer Science Education*, 2014, pp. 33–38.

[37] A. L. D. Buisman and M. C. J. D. van Eekelen, "Gamification in educational software development," in *Proc. the Computer Science Education Research Conference*, 2014, pp. 9–20.

[38] C. R. Prause and M. Jarke, "Gamification for enforcing coding conventions," in *Proc. the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 649–660.
[39] K. Govindarajan, V. S. Kumar, D. Boulanger, and Kinshuk, "Learning analytics solution for reducing learners' course failure rate," in *Proc. 2015 IEEE Seventh International Conference on Technology for Education (T4E)*, IEEE, 2015, pp. 83–90.
[40] K. Berkling and C. Thomas, "Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure," in *Proc. 2013 International Conference on Interactive Collaborative Learning (ICL),* IEEE, Sep. 2013, pp. 525–530.
[41] E. L. Deci and R. M. Ryan, *Handbook of Self-Determination Research*, University of Rochester Press, 2002.
[42] R. van Roy and B. Zaman, "Why gamification fails in education and how to make it successful: Introducing nine gamification heuristics based on self-determination theory," *Serious Games and Edutainment Applications*, 2017, pp. 485–509.
[43] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: classic definitions and new directions," *Contemp Educ Psychol*, vol. 25, no. 1, pp. 54–67, 2000. doi: 10.1006/ceps.1999.1020
[44] PMD. (Oct. 17, 2022). [Online]. Available: https://pmd.github.io
[45] D. V. Bruggen. (Oct. 17, 2022). The most popular parser for the Java language. [Online]. Available: http://javaparser.org
[46] L. YourKit. (Oct. 17, 2022). Java API for creating, reading and updating dictionaries in WordNet format. [Online]. Available: https://extjwnl.sourceforge.net/
[47] O. Karnalim and W. Chivers, "Gamification to Help inform students about programming plagiarism and collusion," *IEEE Transactions on Learning Technologies*, vol. 16, no. 5, pp. 708–721, 2023.