



How Consistent Are Humans When Grading Programming Assignments?

MARCUS MESSER, Informatics, King's College London, London, United Kingdom of Great Britain and Northern Ireland

NEIL C. C. BROWN, King's College London, London, United Kingdom of Great Britain and Northern Ireland

MICHAEL KÖLLING, Department of Informatics, King's College London, London, United Kingdom of Great Britain and Northern Ireland

MIAOJING SHI, College of Electronic and Information Engineering, Tongji University, Shanghai, China

Providing consistent summative assessment to students is important, as the grades they are awarded affect their progression through university and future career prospects. While small cohorts are typically assessed by a single assessor, such as the module/class leader, larger cohorts are often assessed by multiple assessors, typically teaching assistants, which increases the risk of inconsistent grading.

To investigate the consistency of human grading of programming assignments, we asked 28 participants to each grade 40 CS1 introductory Java assignments, providing grades and feedback for correctness, code elegance, readability and documentation; the 40 assignments were split into two batches of 20. The 28 participants were divided into seven groups of four (where each group graded the same 40 assignments) to allow us to investigate the consistency of a group of assessors. In the second batch of 20, we duplicated one assignment from the first to analyse the internal consistency of individual assessors.

We measured the inter-rater reliability of the groups using Krippendorff's α —an $\alpha > 0.667$ is recommended to make tentative conclusions based on the rating. Our groups were inconsistent, with an average $\alpha = 0.2$ when grading correctness and an average $\alpha < 0.1$ for code elegance, readability and documentation.

To measure the individual consistency of graders, we measured the distance between the grades they awarded for the duplicated assignment in batch one and batch two. Only one participant of the 22 who didn't notice that the assignment was a duplicate was awarded the same grade for correctness, code elegance, readability and documentation. The average grade difference was 1.79 for correctness and less than 1.6 for code elegance, readability and documentation.

Our results show that human graders in our study cannot agree on the grade to give a piece of student work and are often individually inconsistent, suggesting that the idea of a 'gold standard' of human grading might be flawed. This highlights that a shared rubric alone is not enough to ensure consistency, and other aspects such as assessor training and alternative grading practices should be explored to improve the consistency of human grading further when grading programming assignments.

Funding support for this article was provided by the King's College London College Teaching Fund.

Authors' Contact Information: Marcus Messer (corresponding author), Informatics, King's College London, London, United Kingdom of Great Britain and Northern Ireland; e-mail: marcus.messer@kcl.ac.uk; Neil C. C. Brown, King's College London, London, United Kingdom of Great Britain and Northern Ireland; e-mail: neil.c.c.brown@kcl.ac.uk; Michael Kölling, Department of Informatics, King's College London, London, United Kingdom of Great Britain and Northern Ireland; e-mail: michael.kolling@kcl.ac.uk; Miaojing Shi, College of Electronic and Information Engineering, Tongji University, Shanghai, China; e-mail: mshi@tongji.edu.cn.



This work is licensed under Creative Commons Attribution International 4.0.

© 2025 Copyright held by the owner/author(s).

ACM 1946-6226/2025/9-ART49

<https://doi.org/10.1145/3759256>

CCS Concepts: • **Social and professional topics** → **Student assessment**; • **General and reference** → *Empirical studies*;

Additional Key Words and Phrases: Computer Science Education, Assessment, Programming Assessment, Consistency

ACM Reference format:

Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaoqing Shi. 2025. How Consistent Are Humans When Grading Programming Assignments?. *ACM Trans. Comput. Educ.* 25, 4, Article 49 (September 2025), 37 pages.

<https://doi.org/10.1145/3759256>

1 Introduction

As many computer science cohorts are becoming larger [10, 37, 40], delivering high-quality grades and feedback for student assignments within a timeframe that maximises learning is becoming increasingly difficult [20, 31]. Promptly providing consistently applied assessment criteria is critical to maintaining student satisfaction [30].

Summative assessment, where the student is evaluated on what they have learnt during the semester or chapter [24], majorly impacts the students' outcomes throughout and after their degree [34]. During their degree, their grades in the previous year's courses can determine the students' progression to the next year. Students' final degree classification or grades are often factors in their employment prospects outside of education [34, 61]. Having processes to ensure consistent or fair marking is crucial to maintaining the perceived value of summative grades and students' perceptions of justice [47].

While small cohorts are often assessed by a single assessor, such as the module/class leader, larger cohorts require multiple assessors, typically graduate teaching assistants, to provide grades and meaningful feedback in a short timeframe. These assessors often use a rubric comprising the evaluation criteria, quality definitions, and a scoring strategy [51] to grade and provide meaningful feedback on the students' work. The rubrics can be written at varying granularities, from generic criteria encompassing the final grades, to individual rubric items allocating individual marks. However, using multiple assessors can lead to issues with how consistently the rubric is applied [46], especially for subjective elements of an assignment like code quality. Though rubrics provide a scoring strategy for teaching assistants, many experience uncertainty when assessing, even those who are experienced teaching assistants [53].

Instead of using multiple graders to assess large cohorts, many large programming courses often use automated assessment tools to grade their assignments. While the automated approaches can provide a consistent grade based on provided criteria, such as unit tests, they often introduce other issues. For example, using automated assessment tools that implement tests requires a stringent structure and a well-tested and developed test suite to produce grades accurately [42]. These test suite approaches often take time to perfect and cannot search for additional marks for students on the grade boundary or grade those that submit uncompileable code. Nor can they grade issues such as code structure or readability.

Neither human grading nor automated assessment tools are infallible. As the grades awarded are of such high importance, many higher education institutions have policies or practices to minimise the impact of mistakes within the assessment. Examples of these policies or practices in our country and our institution specifically include:

- Assessment boards and sub-boards often oversee all aspects of assessment, from the design of the exam or coursework to the marking scheme to the awarded marks.

- Second marking is often used to validate a single marker’s work and can be done blind, without the grades and feedback from the first marker, or they can be provided with the grades and feedback from the first marker.
- Scaling marks, especially if a new exam or coursework has performed significantly lower than previous course iterations.
- Stepped marking is where assessors can only select certain marks steps, such as for a first, 72, 75, or 78 can be selected, limiting issues with students getting final marks at the classification bounds.
- Student appeals are where the students can request a regrade request on coursework if they disagree with a grade or feedback they have been given.

Our study aims to investigate the consistency of human graders when assessing a CS1 programming assignment, including how consistently the rubric is applied between multiple and individual graders, by answering the following **Research Questions (RQs)**:

- RQ1.* How consistently does a group of typical graders apply a grading rubric? Specifically, what is the inter-rater agreement of multiple assessors grading the same submissions?
- RQ2.* Are individual graders consistent at applying a grading rubric? Specifically, do they provide the same marks for the same work when marked a second time?

These questions will have implications for CS1 courses: how consistent is the grading on large courses with multiple markers; how consistent are single graders (relevant to small courses); and how consistent is human grading in general, given that many course leaders will contemplate the use of automated or AI-driven grading?

We pre-registered our study on the Open Science Foundation before conducting the study [3]. Section 2 discusses existing work on consistency of grading practices and grading with groups of assessors. Section 4 introduces our data collection and analysis methodology. Section 5 presents the results from our pilot study and our full study, and Section 6 discusses our results and answers our RQs. We describe threats to validity in Section 7 and conclude our results and discussion in Section 8. Our contributions in this article include:

- A new ‘Menagerie’ dataset: a publicly available graded dataset of students’ assignments, available to the research community, and suitable as a data source for further studies concerning grading accuracy and consistency.
- An in-depth analysis of the consistency of awarded grades when utilising multiple graders.
- A comprehensive investigation into the awarded grades for individual graders when marked the same assessment a second time.

2 Related Work

2.1 Consistency in Grading

Applying assessment criteria consistently with detailed feedback is a key factor in student satisfaction [30]. However, evaluating the consistency of grading and feedback is not often researched. Migut and Wiersma [43] presented a poster showing their preliminary study on whether grading an online exam matches traditional paper grading and found a small but significant difference between online and paper grading. Borela and Roy [8] also presented a poster showing the outcome of a workshop to train TAs with a rubric, which showed a reduction in grade variance after the workshop.

Both Ahoniemi et al. [1] and Auvinen [4] developed rubric-based grading tools to improve grading consistency when grading large classes. Auvinen’s tool, Rubyric, provides an interface for instructors to create rubrics and add feedback phrases. They conducted user satisfaction surveys

and found that the six undergraduate teaching assistants who graded the assignments thought that Rubyruc made them more consistent. Students tended to agree that the feedback was useful and detailed [4]. Similarly, Ahoniemi et al. developed a tool called ALOHA, which allows instructors to construct a rubric and add feedback phrases for common mistakes. They conducted both an instructor survey and a statistical evaluation of the tool. The survey found that the instructors found the semi-automated feedback phrasing particularly useful. The statistical analysis, which compared grades when graded with a paper rubric and those graded with the tool, found that those using the tool had no significant difference between the graders, and when grading without the tool, had a significant difference between the grades given by the most lenient grader and two stricter graders.

Evaluating the consistency of assessment has been conducted in numerous other disciplines. Bloxham et al. [7] took a multidisciplinary approach and investigated 24 assessors from various disciplines: psychology, nursing, chemistry and history. Each assessor was given five assignments to grade, each on a typical task for their discipline. They used Kelly's repository grid exercise [22] to elicit constructs and grades by first comparing combinations of three assignments, stating which two are the same and how one is different, and applying a score between 1 and 5 for these constructs. This study aimed to determine the consistency of the individual constructs, and they found no consistency in the rankings of constructs within any of the subjects.

Within medical education, Dunbar [19] evaluated the consistency of nursing assessment. They ran a study asking seven educators to assess a reenacted physical examination in person and a recording of the same examination a month later. They found that the grading in person and the recording had an inter-rater agreement of approximately 84% between the participants, with the inter-rater measurement calculated by percentage agreement. McManus et al. [41] explored the variance of examiner leniency and stringency within clinical examination. They used 3 years of exam grades, approximately 10,000 candidates, each graded by two examiners, and applied the Rasch model, a latent trait model used to determine the probability of a person succeeding on an item [66]. Their results suggest that greater examiner stringency was associated with greater examiner experience.

Henderson et al. [26] examined the grading practices of 30 assessors when grading exams. The participants, who were from multiple institutions, were provided with five answers to an introductory calculus exam, with two answers being of interest for investigating the conflict between valuing reasoning and correctness. Overall, the scoring of these two solutions differed significantly among individual instructors, with 12 instructors awarding higher grades for one solution than the other, 13 awarding lower grades, and 5 awarding equal grades for both submissions.

Wiley and Gardner [64], investigated methods to ensure marking consistency when using multiple tutors to grade a second-year engineering course in design fundamentals. Each tutor was asked to grade their session tutorial of 32 students, and to achieve a consistent standard of marking between different tutors, they used double-blind marking and remarking. As part of the grading process, the tutors were arranged into small groups to discuss and regrade the assignments collaboratively. The course coordinator also provided two benchmark assignments from previous years that the tutors subsequently graded using specified criteria, effectively reducing the variability in grading between different tutors. While the study found only small variations between markings from different tutors, students still complained of a perceived lack of consistency, specifically being exacerbated by inconsistencies in how different tutors delivered feedback.

Hicks and Diefes-Dux [27] examined the consistency of teaching assistants when grading a first-year engineering course's engineering MATLAB assignments. The course was split into 15 sections, with five undergraduate and one graduate teaching assistant, where the undergraduate teaching assistants graded the homework assignments while supervised by the graduate teaching assistant. The undergraduate teaching assistants were asked to grade against a rubric, with most criteria having four achievement levels: no evidence, under-achieved, partially achieved and fully

achieved. As a baseline grade, the course leader graded 172 submissions. The authors utilised percentage agreement, Cohen's kappa and Krippendorff's alpha to measure the consensus and consistency between the course leader and the undergraduate teaching assistants. They found that the course leaders and undergraduate teaching assistants had an agreement of 49.4% and that all metrics indicated that the undergraduate teaching assistants were applying the same interpretations of the rubrics that the course leaders intended.

Passonneau et al. [49] as part of their investigation into whether analytic grading rubrics, which have more specifics than the rubrics provided to the students to define expectations, produce reliable assessments, they analyse the inter-rater reliability of four undergraduate teaching assistants when grading physics lab reports. They found that grades administered by the teaching assistants did not correlate to *post hoc* assessments from trained raters, identified missed learning opportunities for students and that instructors were misled about student progress.

While investigating the consistency of the grading process is conducted within other domains, from the best of our knowledge, this research is rarely conducted within computer science education. Multiple and individual graders consistently applying a grading rubric is critical in providing a fair and meaningful assessment of students' work. In turn, it improves a student's learning and overall satisfaction with the course.

2.2 Rubric Design

Rubrics are often used to make assessment expectations transparent [11], and they describe desirable qualities and common pitfalls in student work against a gradation of quality [2]. A rubric has three essential features: the evaluation criteria, the definitions of quality and a scoring strategy [51].

A review by Reddy and Andrade [51] discussed rubric use in higher education. They present 20 articles on rubrics and include topics on student and instructor perceptions, use and academic performance, and rubrics' validity and reliability. They found that instructors can have contrasting perceptions of using rubrics, with some believing that rubrics provide an objective basis for evaluation, while others resist integrating rubrics into their assessment. The review also investigated how a well-designed scoring rubric should alleviate inconsistencies in the grading process, both when rubrics are used by a single grader or a group of grader. A well-designed rubric should minimise errors due to grader training, grader feedback, and clarity in the criteria description. They further discuss how there is ample evidence of disagreement between assessors when using rubrics and stress the importance of grader training.

Jönsson and Svingby [29] reviewed the reliability, validity and educational consequences of using rubrics in assessment. They analysed 75 studies and found that rubrics can improve the reliability of assessment if they are analytic, topic-specific, and complemented with exemplars or rater training, especially with open-ended assessment, as it is not always possible to limit the format of the assessment to achieve high levels of reliability without sacrificing the validity.

Rinne [56]'s recent study aimed to identify sources of inconsistency in assessing undergraduate dissertations from a primary school teacher education programme. They identify that rubrics are often short and open to interpretation, resulting in inconsistent assessment, and discuss sources of inconsistency when grading, which include examiners' interpretations of quality, and found that the same grade could be awarded for different reasons, and the same judgement could result in different grades.

To increase the agreement in teachers' grading, Jönsson et al. [28] conducted a study into whether analytic or holistic rubrics enabled consistent assessment. Analytic assessment involves assessing different aspects of student performance, such as mechanics, grammar, style and organisation. On the other hand, holistic assessment means making an overall assessment that considers all criteria simultaneously [28]. Jönsson et al.'s study investigated analytic and holistic grading in both an

English as a Foreign Language (EFL) and a mathematics high-school course. They found that analytic grading is slightly preferable to holistic assessment regarding teacher agreement for both EFL and mathematics. They further conclude that the results support the argument that teachers' grading is 'complex, intuitive and tacit' [7] and that any attempts to achieve grading consistency are likely futile. However, their results also indicate a small step towards increasing agreement.

In addition to the active area of research investigating how to design rubrics to maximise consistency while maintaining validity, within computer science, defining rubrics for aspects of programming, especially code quality, is still a developing field. Stegeman et al. [60] defined a code quality rubric for introductory programming, including names, comments, formatting and expressions. They tested this rubric against several programming assignments and found that no substantial parts of code quality appeared to be missing. Recently, Kirk et al. [33], motivated by the difficulty for programming educators to find appropriate guidance on teaching style concepts, have defined a literature-informed model for code style, which includes many of the same topics introduced by Stegeman et al. but provides further depth and rationale behind the elements. The model provides an abstract level of the main style aspects that map to understanding and changing code and include principles that relate to language, layout, constructs, design, redundancy, implementation, repetition and structure.

2.3 Teaching Assistants and Assessment

Teaching assistants are often crucial in supporting students' learning; they typically consist of undergraduate, master's and PhD students [54, 55, 63].

The three most common duties for teaching assistants include assisting in programming labs, leading sections to reinforce concepts, and grading; many institutions utilise undergraduate and graduate teaching assistants [44, 63, 64].

Mirza et al. [44] conducted a systematic literature review into how undergraduate teaching assistants are used within computer science courses and found that UTAs often grade assignments, whether they provide the final grades for the assignment, assign coarse primary grades or are one of the multiple undergraduate teaching assistants to grade each student assignment.

Riese et al. [55] explored the challenges faced by teaching assistants in computer science education across Europe. At all the institutions involved in their analysis, teaching assistants were involved in assessment in some form, whether that be assigning points or pass/fail grades. They found that teaching assistants often faced challenges when assessing student work, including understanding the assessment criteria, providing meaningful feedback and failing students.

Wald and Harland [63] discuss the trade-offs of employing student teaching assistants in their position paper, including how they can reduce teaching loads for academics and provide graduate students with teaching experience and financial support. However, teaching experience can differ between teaching assistants, especially undergraduate teaching assistants, who typically have limited academic and pedagogical knowledge, which are both essential for student learning. They suggest limiting the role of graduate and late-stage undergraduate teaching assistants as graders, tutors, and demonstrators, as they have yet to find any sound pedagogical arguments supporting undergraduate teaching assistants as graders. Instead, they suggest that they facilitate discussion and review course material.

Kristiansen et al. [36] also highlight the importance of properly trained teaching assistants in their study comparing manual and automated feedback. They found in a controlled study of undergraduate students in a programming assignment that there was a need for both automated and manual feedback, providing superior results in task effectiveness and student preferences than just automated or manual assessment.

Riese and Bälter [54] conducted a qualitative analysis into how course coordinators use assessment throughout their course and what they expect from TAs. They found that the seven experienced course coordinators in CS1 courses typically use TAs to provide continuous assessments for large cohorts. They discuss how some TAs set the bar too high while others set it too low, which can lead to inconsistency and that some TAs can view their students as friends, leading to ethical dilemmas. Similarly, to Kristiansen et al., Riese and Bälter suggest that not all tasks are suitable for TAs and that assessing students' competency and knowledge can be difficult for experienced teachers.

2.4 Educational Datasets

For our study, we required a dataset of student submissions for the grader participants to grade. While several datasets of students' programming assignments exist, many are not publicly accessible [42], consist of small-scale programs [5, 16] or do not capture contextual details [12].

Some project-scale datasets exist, such as the Scratch website scenarios¹ or the Blackbox dataset [12] that are public or available on request. However, they have the disadvantage that the context of the users is unknown (they may be students at any stage of education or not even students at all), and the aim of their project is unknown, preventing judgment against any criteria for progress or success.

In smaller-scale datasets, including FalconCode [16], Hour of Code², Dublin City University's Programming Submission Dataset [5], and online judge tools, such as HackerRank³, the task is known. However, all of these datasets are closed-ended and focus on small-scale tasks, including highly structured template code, specific names of classes and functions defined in the requirements, or simple one-function tasks, such as `FizzBuzz` or the Rainfall Problem [59]. Therefore, we would not be able to evaluate the graders consistently when evaluating aspects of code quality, as many of the students' decisions regarding code design are already made for them, and many of these datasets use documentation to explain the task.

To the best of our knowledge, no large project-scale publicly available datasets exist. Therefore, we developed our dataset, Menagerie. It consists of real CS1 Java programming submissions for a simulation-based assignment and the associated grades and feedback that arose during this study; more details can be found in Section 3.1.

3 Research Context

3.1 The Menagerie Dataset

As discussed in Section 2, most publicly available programming datasets do not contain detailed grading or feedback data. If they contain grades, they are often limited to whether the program passes or fails a set of automated tests, as most automated assessment tools utilise unit testing to assess the submissions [42]. As such, we developed our own publicly available programming dataset containing human graders' assessments.

We introduce the Menagerie dataset, which comprises 667 real student submissions for King's College London's second semester CS1 Introduction to Programming Java course and is available publicly, see Section 9. The same assignment was given to students between 2017 and 2021 inclusive, with minor changes each year. The changes each year included updating the assignment instructions to clarify points of misunderstanding from the previous years' assignments, with the tasks and the provided template code remaining the same year on year.

¹Scratch Scenarios: <https://scratch.mit.edu/ideas>.

²Hour Of Code: <https://code.org/research>.

³HackerRank: <https://www.hackerrank.com/>.

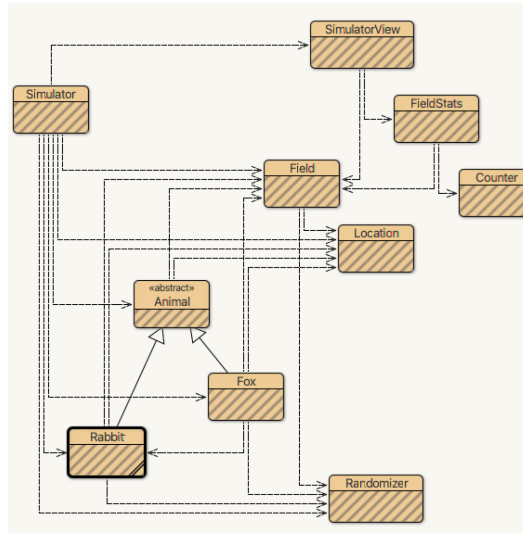


Fig. 1. The UML diagram of the provided template code.

The dataset consists of the following:

- The assignment requirements and grading rubric.
- The project’s starter code.
- The students’ Java files.
- Grades and associated feedback for 272 submissions (produced by the current study).

The dataset does not include the students’ original awarded grades, as we did not receive ethical approval to release them publicly. However, as part of the current study, we asked seven groups of four to grade and give feedback on 40 submissions per group. We produced grades and feedback for 272 submissions; seven assignments were duplicated to evaluate grader self-consistency. Details about the grades and feedback can be found in Section 3.2.

We have received ethical approval from our institution⁴ to conduct this study and publicly release the summary of the grader demographics and the individual grades and feedback from each grader.

3.1.1 Assignment Outline. The assignment was a small-group, open-ended paired programming assignment to utilise object-oriented programming concepts to develop a predator/prey simulator with groups of two or three. The students were asked to conduct pair programming with explicit instructions on how to pair program. One student was the ‘driver’, who wrote the code, and the other was the ‘navigator’, who reviewed the code as it was written, and they were instructed to switch roles throughout the assignment. The students were provided with a template project based on the ‘foxes-and-rabbit’ project from the sixth edition of Barnes and Kölling [6]’s book on ‘Objects First with Java’. The template includes a graphical user interface, a Field class, which contains a two-dimensional array for the simulation environment, and two animals, a Fox and a Rabbit; Figure 1 shows the class diagram of the template code. Students were asked to extend the template code with the following base tasks:

- The simulation should have at least five species, with at least two being predators and at least two not being predators.

⁴Ethics review reference: LRS/DP-22/23-35578.

- At least two predators should compete for the same food source.
- Some or all species should distinguish between male and female animals and only propagate when male and female species are in a neighbouring cell of the two-dimensional array.
- The simulation should keep track of the time of the day, and some species should exhibit different behaviours at some time of the day.

After completing the base tasks, the students were asked to implement one or more challenge tasks. The students could choose to invent their own tasks or to use one or more of the following suggestions:

- Simulate the lifecycle of plants, including growth and being a food source for at least one animal.
- Simulate changing weather states and how they affect other simulation aspects.
- Simulate disease within the species, including the spread of the disease.

During the assignment, students could choose which species they implemented, how those interacted with other species, and how their implementation completed the provided tasks. These opportunities to decide what they implemented allowed students to apply their creativity, choose how they designed their code, implement object-oriented principles, and name classes and functions. The ability of the students to design their solution to a problem and the open-ended nature of the assignment allows instructors to assess a wider variety of skills than they typically could in a close-ended small-scale assignment like the Rainfall problem [21, 59]. In this assignment, the students were assessed on the correctness of the implementation, how elegantly they implemented the solution, the overall readability of the source code, and how well they documented their code. A detailed example submission and the details on the data processing pipeline for cleaning the dataset can be found in Appendix A.

3.1.2 Dataset Limitations. The students were asked to submit a report discussing their implementation as part of the assignments. However, we have excluded these from the dataset, and they were not used to grade the assignments, as they cannot be suitably anonymised for public release. While we automated the de-duplication of the group submissions and removed further duplicate submissions during the anonymisation, there is a remote chance that some repeated assignments within the dataset could still exist.

While the students were given instructions on how to pair program, the assessment did not capture if and how the students conducted pair programming. Anecdotally, some groups started pair programming, especially in the associated lab sessions. However, as the submission deadline approached, groups tended to separate the work, with both students writing code in parallel.

As these are historical assignments that we received *post hoc* permission to release, the students' demographics were not captured. However, all students were in their first year at King's College London and were a mixture of domestic and international students.

3.2 Assessment

The participants graded the submissions from the Menagerie dataset, as discussed in Section 3.1, and used the rubric from the course as guidelines on how to assign grades, discussed in Section 3.3 and provided in Appendix B. The grades and feedback were captured using Gradescope [58], a commercial grading platform designed for STEM assignments, including programming assessments, and allows for rubrics to be uploaded and for graders to provide line-level or overall feedback. While we captured the feedback, which is available in the dataset, this study focuses on the grades provided.

3.3 Rubric

The rubric we provided participants can be found in Appendix B. This is the original rubric from the assignment from which the submissions were captured. The participants were asked to supply letter grades between A and F, with '+' and '-' grades for all letters; for exceptional submissions, participants could award A++. The rubric in Appendix B includes number bounds for each letter; this is due to Gradescope only supporting numerical grades. Within Gradescope, we supplied options that corresponded to the letter grades, with '+' and '-'. The numbers were not used for this study, and participants were instructed to select letter grades in Gradescope.

Participants were asked to grade the following programming skills:

- Correctness. Assess if a student has comprehended and completed the assignments in a way that complies with the assignment requirements [23].
- Code Elegance. Investigate the overall code design. Have the students use Java design principles correctly, including inheritance and polymorphism, methods to reduce duplicated code, and their overall approach to implementing the required features [9, 33].
- Readability. Evaluate whether a student's submission is easy to understand. This can include code style, whether formatting, indentation and naming conventions [9, 33].
- Documentation. Assess if and how well students have documented their code, including inline comments and the natural language code summary (docstring) [9, 33, 50].

While code elegance, readability, and documentation are all aspects of code quality, we asked participants to provide grades for each aspect, allowing for richer data capture of different aspects of code quality. Similar aspects have been utilised in multiple different studies [42, 48]. Typically, these three aspects of code quality and correctness would be aggregated into a single final grade; we chose not to aggregate the grades, as different assignments will provide different weightings on these aspects. Thus, we analyse the consistency of grading code elegance, readability, documentation, and correctness, individually. We leave the methods of combining the grades for educators, but it does not affect this investigation into the consistency of grading the constituent components.

This rubric consists of the three essential features of a rubric, as discussed in Section 2.2, including the evaluation criteria, the definitions of quality, and a scoring strategy. While some improvements could be made to the clarity of the definitions of quality and the scoring strategy, the rubric we provided our participants was the original rubric from the course, which provided the following:

- A typical representation of a rubric, which has seen many years of use at King's College London.
- The same rubric that historical students provided alongside the assignment requirements guided the design and development of their submissions.

4 Methodology

4.1 Group Consistency (RQ1)

To explore the consistency between graders, each assignment was graded four times, each time by a different grader. Figure 2 shows an example of what the data includes after the grading is complete. Grading each assignment multiple times by different graders allowed us to investigate how consistently a group of graders apply the grading rubric to individual assignments.

To investigate how consistently a group of graders apply a grading rubric, we used Krippendorff's α (implemented using the Fast Krippendorff library [13]) to assess the reliability of agreement between the graders on the awarded grades. Krippendorff's α is a percent chance-based reliability coefficient, similar to Cohen's κ and Fleiss' κ . It evaluates the deviation from perfect reliability

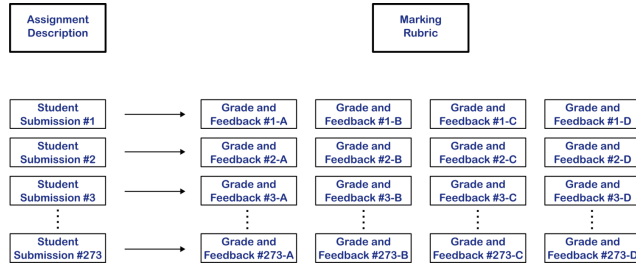


Fig. 2. The data includes an assignment description, a marking rubric, a set of student submissions attempting the assignment, plus a set of grades and feedback from multiple human markers (A–D here, although more markers are involved) for each student submission.

by the proportion of observed to expected disagreement and allows any number of raters and categories while allowing for missing data [35]. It also has several weightings that can be applied to different types of data, including ordinal, which is data that is both categorical and has an order, such as letter grades.

As many percent chance agreement-based coefficients, including Krippendorff’s α and Fleiss’ κ , underestimate the agreement when one of the labels is dominant [18, 25, 65], we also compute Gwet’s AC_2 , which is a weighted variant to Gwet’s AC_1 , which was designed to be robust in the high prevalence of a single class [25]. Gwet’s AC_2 is more robust in instances of high prevalence, as it removes from consideration all agreements that occurred by chance [25].

As we planned, each assignment was graded by four of the graders rather than all graders, making Krippendorff’s α and Gwet’s AC_2 the ideal measures. Using Krippendorff’s α and Gwet’s AC_2 varies from our pre-registration, as our original choice of metric (Fleiss’ κ) required a complete set of grades and did not consider the ordinal nature of grading, so we realised that this was not the best choice.

4.2 Individual Consistency (RQ2)

To aid in the research into self-consistency and the study pacing, we supplied the graders with batches of submissions, allowing us to repeat a few chosen submissions for the graders to regrade, resulting in some submissions being graded eight times. We chose to repeat assignments representative of the dataset, specifically assignments of the median size, a common theme, and submissions that fall near the mean of the grades awarded within the previous batch. The original assignment for Menagerie was a pair-programming assignment, which was de-duplicated; however, duplicates could still exist and we told our graders this as a cover story for why they might see multiple similar assignments. The possibility of duplicates, specifically choosing assignments and the time between batches, helped mitigate the likelihood of the graders noticing that they were grading the same assignment (which might have led to the graders going back to the first grading attempt and copying and pasting their grades instead of grading it fresh a second time).

To evaluate the consistency of individual participants, we calculate the distance between the grade awarded in the first and second batches for each participant and for each skill. We could not use typical inter-rater analysis to evaluate self-consistency, as the two grades are too few for inter-rater reliability metrics, including Krippendorff’s α , Gwet’s AC_2 and Fleiss’ κ . We chose to limit the number of repeated assignments to ensure that the participants were unaware that they were grading a duplicate and to maximise the number of annotations for the dataset.

To provide further information on the participants’ experience grading these assignments, we conducted a semi-structured interview. These interviews were conducted in English by one of the

members of the research team and lasted approximately 30 minutes; Microsoft Teams was used to provide automatic transcription. During the interview, the participants were asked the following questions:

- (1) Can you tell me about your experience grading these assignments?
- (2) Can you tell me about your grading process?
- (3) How well do you think the rubric aided you when grading the assignment?
- (4) Did you review your grades to validate your consistency?
- (5) How do you think the environment you were in while grading affected your grading?
- (6) How do you think the mood you were in while grading affected your grading?
- (7) How do you think the time of day you graded per session affected your grading?
- (8) Were there any specific submissions that stuck in your mind?
- (9) Did you notice any duplicates or near-duplicates in the data?
- (10) Do you have any further comments on your experience?

Unfortunately, many participants did not provide detailed answers for most of the questions asked, even when prompted for more information. This is likely due to the interviewer's lack of experience conducting semi-structured interviews. As such, we have decided not to provide an in-depth analysis of the semi-structured interviews and only use the answers from questions four and nine in our results. These questions were coded as 'yes' or 'no' by one member of the research team and used to provide further context to our analysis of participant's individual consistency.

4.3 Pilot Study

To validate our data collection process, we conducted a pilot study with two assessors: one who was a final-year computer science PhD student with numerous years of grading this specific assignment and a first-year computer science PhD student with 1 year of grading experience and had previously completed this assignment as an undergraduate. They were provided the pre-study survey, asked to grade 2 batches of 20 assignments each, and given 2 weeks to complete each batch, with a 2-week break. After grading, both participants were asked to attend the post-study semi-structured interview, which we expanded for the pilot study to ask for any improvements that could be made for the full study.

The participants completed the grading within the supplied time frame and found Gradescope helpful when grading. However, they both struggled to remember to complete the grading diary and requested more information about the assignment to be supplied.

For the full study, we provided detailed instructions on using Gradescope, a brief overview of what the students had learned before submitting the assignment, the coursework specification, rubric, and template code. We added a link to this information and the diary to each submission in Gradescope, so that participants were automatically reminded at the end of grading each submission.

4.4 Participant Selection

We conducted a pre-study survey to identify suitable grader participants and collect demographic data. Our criteria for our participants follow the typical demographics of teaching assistants performing grading in many institutions [44, 55] and are as follows:

- Students at our institutions in their third year of undergraduate, a master's student or a PhD student.
- They identify as proficient in Java programming and have at least 3 years of programming experience.

All selected participants were remunerated £280 GBP (\$340 USD) for participating; this rate was concurrent with our institution's hourly rate for teaching assistants. We decided it was important that our participants receive pay similar to what they would receive when grading for a live assignment to ensure they take the study seriously and complete all their grading.

4.5 Position Statement

The researchers in this study are three academic staff members and a graduate student at a large UK university. The dataset is captured from one of the academic staff's courses, CS1 Java courses that ran between 2017 and 2021 inclusive, with permission from the college's ethics committee (LRS/DP-22/23-34272), the students' permission was not required for this data, and all data was validated by research assistants to ensure the data was anonymised, as prescribed by the college's data handling policy. The participants who took part in the study are current students at our institution but are not current students of any members of the research team. The consistency study was further approved by our college's ethics committee to use the existing dataset and to provide remuneration to our participants (LRS/DP-22/23-35578).

5 Results

5.1 Participant Demographics

Through our pre-study survey, we selected 28 participants who matched our criteria to complete the study. Twenty-five of our participants had over 5 years of programming experience, with three having between 3 and 5 years, and all identified as proficient in Java. Seven participants were PhD students, 13 were Masters students, and 8 were third-year undergraduate students. Two participants graded for between 2 and 3 years, one participant graded between 1 and 2 years, six participants graded for less than 1 year, and 21 participants had not been assessed before. These are a relatively typical profile of those recruited to act as graders at our institution, and as described in Section 2.3, typical of graders at many other institutions as well.

Most of our participants identified as male, with three identifying as female, which follows a similar distribution of students enrolled in the computer science courses at our institution.

Figure 3 shows the degree level composition for each group of four. Nearly every group contained one or more PhD students, over half the groups contained two or more master's students, and most groups had one or two third-year students.

5.2 Grade Overview

Figure 4 shows the grade distribution for each skill. Most assignments achieved high grades, with approximately 10% achieving below a B-. Those graded F for correctness include assignments that failed to compile. Some participants failed to select a grade for certain skills within certain submissions; these have been omitted from our results. While there is not a lot of variance in the data, anecdotally this is a similar distribution of grades that we saw while conducting the original summative grading.

5.3 Grading Consistency of Multiple Graders (RQ1)

5.3.1 Inter-Rater Metrics. Each batch of 20 assignments was graded by a group of four graders. We discuss our inter-rater reliability results, using Krippendorff's Alpha (α) for each group of four graders and each skill, the spread of the awarded grades, and the reliability of individual grades.

We opted to use the ordinal metric within Krippendorff's α , as letter grades are inherently ordinal; they are categorical and have an order. The ordinal metric applies weights to the α calculation by utilising the ordinal difference between ranks [35].

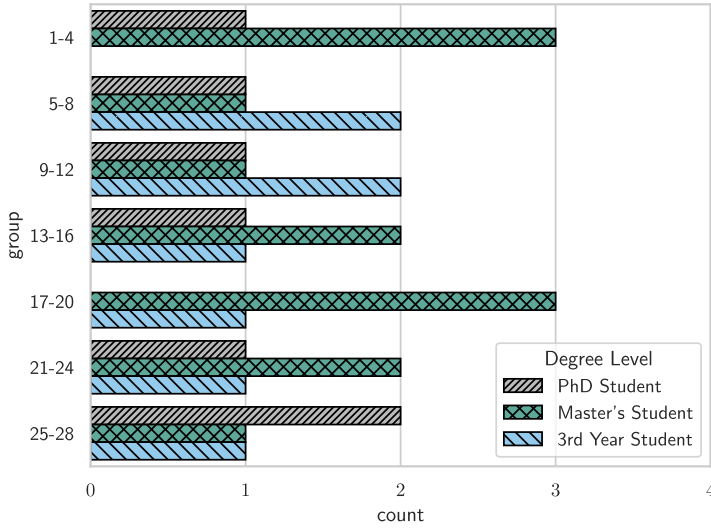


Fig. 3. The degree level composition for each group of participants.

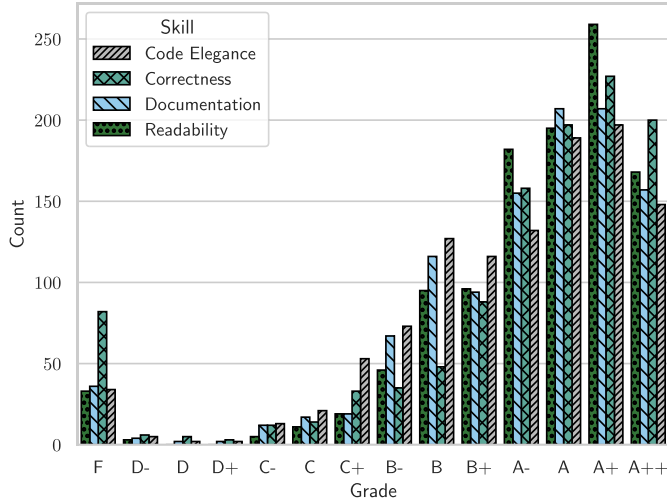


Fig. 4. The count of awarded grades.

When using Krippendorff's alpha, if the raters completely agree, $\alpha = 1$; if there is no agreement, then $\alpha \leq 0$. Krippendorff also suggests that $\alpha > 0.800$ is acceptable and $0.800 > \alpha \geq 0.667$ are only for drawing tentative conclusions [35].

Due to an accidental duplicate when setting up the second batch, participants 21–24 graded assignment 105 twice, these grades have been omitted from our results.

Figure 5 shows the α for each of the seven groups and each skill. These results show that in our study, consistency was low, but were slightly more likely to agree on the grades for correctness and code elegance but did not agree on the grades for readability or documentation, with the average α being 0.220, 0.099, 0.046, and 0.055, respectively.

Figure 6 shows the results of Gwet's AC_2 . Following the same bounds defined by Krippendorff, no groups achieve acceptable agreement ($AC_2 > 0.800$) for any of the skills evaluated. However,

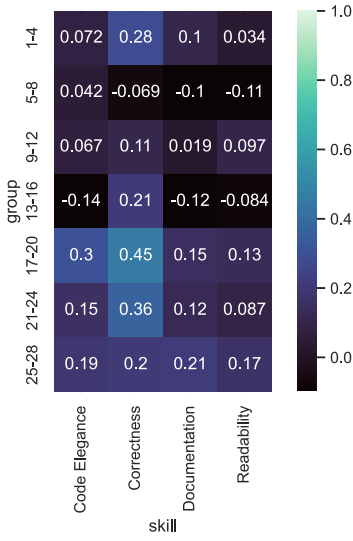


Fig. 5. This heatmap shows the consistency, measured by Krippendorff's Alpha (α) with the ordinal metric, for each group of participants and each skill. Grades Awarded (A++, A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F, Not Graded).

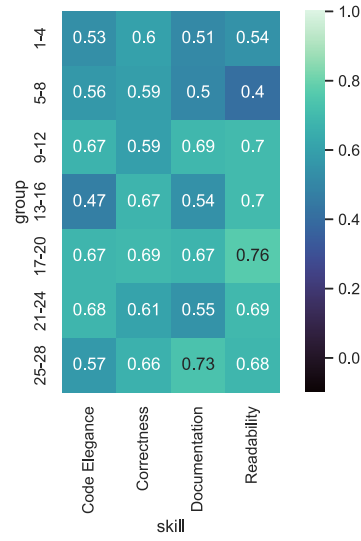


Fig. 6. This heatmap shows Gwet's AC_2 with the ordinal metric applied for each group of participants and each skill.

some groups show partial agreement with $AC_2 \geq 0.667$ for specific skills, though on average, no skill had an $AC_2 \geq 0.667$, with the average AC_2 for correctness being 0.629, for code elegance being 0.592, for readability being 0.638 and for documentation being 0.596.

Out of the four skills that the participants graded, correctness was most likely to be graded more consistently, with five of the seven groups achieving a higher α than the other four skills, with groups 5–8 and 25–28 achieving the second most consistently graded skill for their respective groups. In comparison, all participant groups were similarly inconsistent when grading code elegance, readability and documentation, all skills achieving an average $\alpha < 0.1$.

5.3.2 Grading Experience. Our participants had varying grading experience levels, which could have affected the consistency of their grading. Figure 7 shows the distribution of Krippendorff's α for each of the four skills. Independent of the years of experience, no groups achieved an agreement close to the threshold for partial agreement. Our results for 1–2 years and 2–3 years of experience show varying levels of agreement, as the sample size of participants who had more than 1 year of grading experience was small, which is typical at many institutions where undergraduate, master's and PhD students all assess student work [44], as many students typically graduate after 1 or 2 years of working as teaching assistants. Those with 1 year of experience in grading show no significant improvement in inter-rate agreement compared to those with no grading experience.

Group 17–20, which had the highest average $\alpha = 0.257$ consisted of three master's students and one third-year student. Another group with a similar composition was group 1–4, consisting of three master's students and one PhD student; however, they had an average $\alpha = 0.123$, the fourth lowest average α .

Group 25–28, which had the highest proportion of PhD students, at two, with one master's and one third-year, had the second highest average $\alpha = 0.192$. Those that primarily consisted of third-year students, at two, with one master's and one third year, performed the worst in terms of

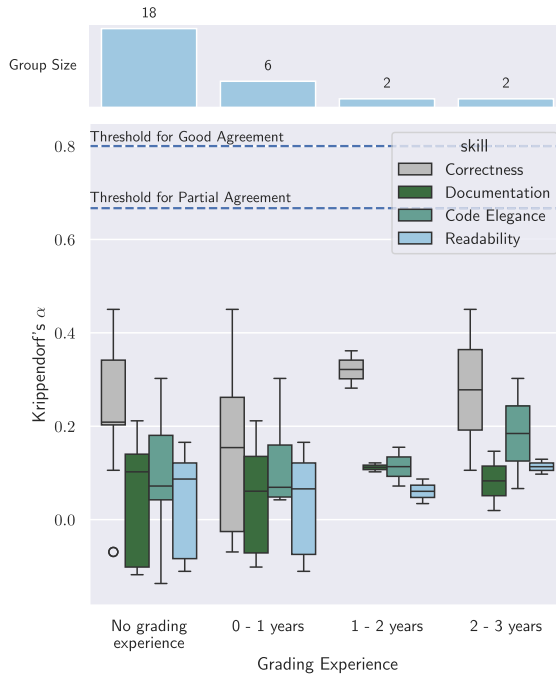


Fig. 7. This figure shows the distribution of Krippendorff's α for the participant's grading experience and all four skills. The box plots' box shows the inter-quartile range, while the whiskers extend to points within 1.5 of the inter-quartile range. The thresholds for good and partial agreement are the thresholds provided by Krippendorff [35]. The bar plot shows the proportion of participants with varying levels of grading experience.

inter-rater reliability, with Group 13–16 having an average $\alpha = -0.032$ and Group 5–8 having an average $\alpha = -0.0598$. In most cases, being of a higher degree level when grading CS1 introductory programming does not increase the consistency when using multiple graders.

5.3.3 Grade Range. To further explore the inter-rater reliability, we evaluated the spread of grades awarded by each group. Figure 8 shows the minimum, maximum, and mean correctness grades for group 17–20, which was the most consistent when grading correctness, with an $\alpha = 0.45$, and for group 5–8, which was the least consistent when grading correctness, with an $\alpha = -0.069$. Appendix C shows the distribution of grades for all groups and all grades.

For our most consistent group (group 17–20), only assignments awarded an F (assignments 80 and 98)—since they either did not compile or did not include any student code—were assessed with complete agreement. The original authors of assignment 80 submitted the template code they provided, and assignment 98 did not compile or, if the assessors fixed the compilation error, had an unhandled `NumberFormatException`, causing the simulator not to execute.

To further investigate the reliability when awarding each grade, we stipulate that assignments grade ranges less than or equal to two (e.g., A to B+ is two grades away, and A to A- is one grade away) have good reliability; those with a range of three (e.g., A+ to B+) have mediocre reliability, and those with a range greater than three (e.g., A- to C+) have poor reliability; where the grade range is the maximum awarded grade minus the minimum awarded grade for each assignment.

Figure 9 shows the proportion of the reliability for each grade and skill and shows that our participants were more reliable at awarding F and D- grades across all four skills than any other,

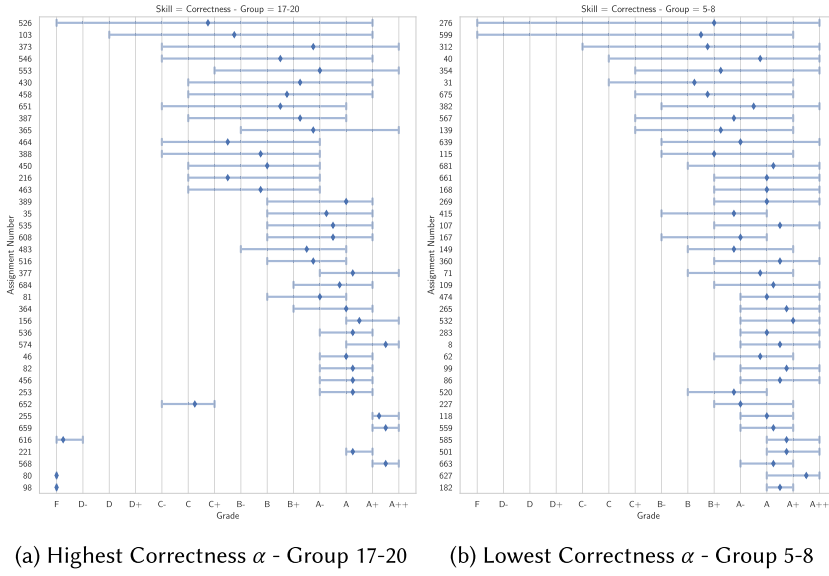


Fig. 8. These figures are for the highest and lowest α and show the minimum, maximum and mean correctness grade awarded by the participants for each assignment. Plots for all groups and all grades can be found in Appendix C.

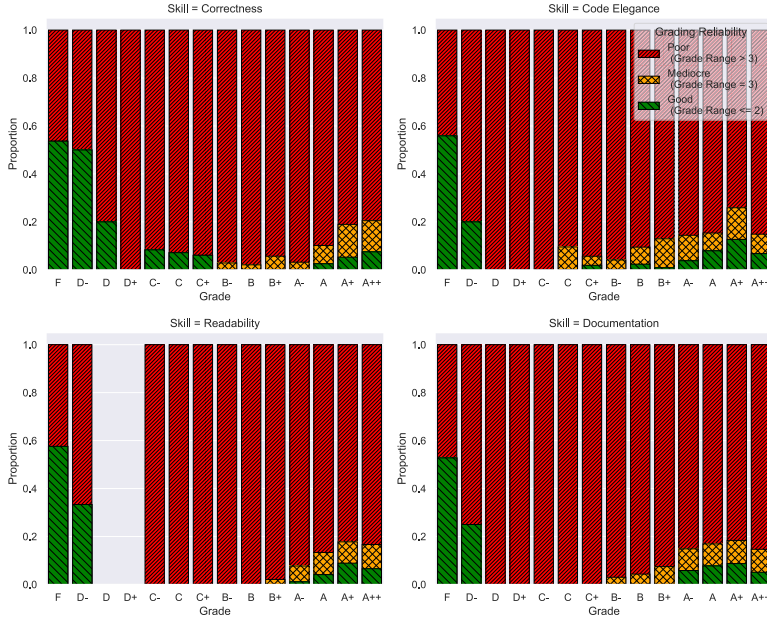


Fig. 9. This figure shows the proportion of how reliable each grade is for each skill across the grading scale. Reliability is defined by the grade range for each assignment, with 'Good' as a grade range ≤ 2 , 'Mediocre' = 3, and 'Poor' ≥ 3 .

with the middle grades (C- to B+) being the most unreliably graded across all four skills. The higher grades (A- to A++) were reliable or partiality reliable in less than 20% of assignments graded across all four skills.

5.3.4 An In-Depth Analysis of the Grades Awarded by Group 17–20. Several assignments were awarded grades that were one grade bound away, with assignments 568, 211, 659, and 255 all being awarded an A++ or an A+ and assignment 616 being awarded a D- or an F. With these assignments being at the tails of the potential grades, it can be difficult to differentiate between the two furthest bounds, especially with the subjectiveness of open-ended assignments. For assignment 616, three of the four assessors graded the assignment an F for correctness, as the submission only contained the base code. However, participant 18 opted to assign the grade D- and gave the feedback, ‘The submission does not cover any of the requirements in the coursework’. We do not know why Participant 18 opted to grade assignment 616 with a D- and assignment 80, another assignment that did not meet the coursework requirements, with an F. However, this does highlight the potential issues with the internal consistency of human graders, which we discuss in more detail in Section 6.2.

Assignment 526 was the most inconsistently graded within the group 17–20, with grades ranging from A+ to F, with two awarding the assignment F for failing to compile, one awarded the assignment an A, and the other awarding the assignment an A+. As part of the anonymisation process, some base classes were copied into the submission. This was brought to our attention early within batch one; all participants were instructed to delete the problem classes and regrade any assignments that could not be compiled. However, some may not have completed this step.

The second most inconsistently graded assignment was 103, with the four participants awarding a D, a C-, a B+ and an A+. Participants 19 and 20, who awarded the assignment a D and a C-, respectively, highlighted that the assignment spawned three windows, with one having no visible functionality, but completed all the base tasks and some of the challenge tasks. Participants 17 and 18, who awarded the assignment a B+ and an A+, respectively, chose to focus on completing the challenge tasks and decided not to penalise the submission for spawning three windows as heavily. Participant 17 stated in their feedback, ‘Whilst the base tasks were correctly implemented, the challenge tasks were incorrectly configured...’. This is an excellent example of the hawk and dove effect, where some examiners are more stringent and require higher performance, the hawks, than others, the doves [41]. In this case, participants 19 and 20 are the hawks and require the program to have minimal errors, while participants 17 and 18 are the doves, and chose to focus on the highlights of what the student has submitted and provide a more lenient grade by overlooking the bugs within the application.

While correctness is mostly objective, code elegance, readability, and documentation are inherently subjective. Even if aspects of these are defined in the course, such as the specific code style or code design practice that the students must use, the examiners have their personal preferences on what makes code readable, well-designed or well-documented. In this assignment, students were asked to use the code style demonstrated in Barnes and Kölling [6] ‘Objects First with Java’ and to use object-oriented programming, including inheritance and polymorphism, which they covered in class in the few weeks before the coursework was set.

Out of our seven groups, group 17–20 had the highest α while grading code elegance with an $\alpha = 0.3$, less than the tentative acceptable value of $\alpha > 0.667$. When grading code elegance, Group 17–20 did not agree for any of the 40 assignments they graded. Two assignments, 568 and 616, out of the 40 that group 17–20 graded were one grade bound away. Assignment 616 was awarded the same grades as correctness, with three awarding an F and the fourth awarding a D-. Assignment 568 was awarded grades at the opposite end of the spectrum, with three assessors awarding an A+ and one awarding an A. The assessor that awarded an A gave the feedback that ‘... some functions are very long...’, whereas two of the assessors who awarded the A+ praised the student for their implementation, while the third did not provide any feedback with their grade.

The largest variance in grades awarded by group 17–20 for code elegance was for assignment 458; the grades awarded were C, B-, A- and A++. Participant 17, who awarded the A-, gave the

feedback ‘Solid implementation on both core and challenge class’. Participant 20, who was awarded the C, commented on the repetitiveness of the code and how functions could be refactored to the superclass. These differences in awarded grades and the associated feedback can suggest how grading subjective elements of assignments can introduce a higher variance in the awarded grades.

This trend of a few assignments being somewhat consistently graded and many being inconsistently graded holds for readability and documentation, with Appendix C showing the grade range for each skill. Figure 9 shows this trend for each grade, with the middle grades (D–B+) more likely to have a larger variance in awarded grades across all four skills. While the middle grades showed no cases of any grading achieving acceptable levels of consistency, the high (A++ to A) and low ranges (F–D-) displayed at least some instances of grader agreement. Grades F and D- have the largest portions of grade ranges, being less than or equal to two. At the other end of the grade spectrum, grades A, A+ and A++ have a smaller proportion of grade ranges that are less than or equal to two.

5.4 Grading Consistency of Individual Graders (RQ2)

5.4.1 Grade Range. As part of our study, we purposefully duplicated an assignment within the second batch to allow us to evaluate the consistency of the individual graders. 22 participants did not notice the duplicate in the second batch; six participants (1, 2, 10, 11, 20, 25) did. Figure 10 shows the distance between the 14 awarded grades when regarding the same assignment for a second time, for each participant for each skill.

The average absolute distance between grades when regarding the same assignment a second time was correctness = 1.786, code elegance = 1.250, readability = 1.357 and documentation = 1.571. These results show that for the consistency of the individual graders in our study, on average, they were between one and two grades apart between batches one and two, with a higher variability in correctness and documentation, and were more consistent with themselves when grading code elegance or readability.

5.4.2 Post-Study Interview. During the post-study interview, 24 out of 28 graders stated that they graded consistently. However, our results show that when grading a repeated assignment at a later period, only two participants graded the duplicated assignment the same across all skills. Only two participants, three and ten, provided the same grade in both batches for all four skills. Participant 10 noticed the duplicate, and during their post-study interview, participant 10 mentioned that they noticed the duplicate and changed the grades for both batches to F as they believed it was plagiarism, while the other five participants followed the instruction that if you notice a duplicate, treat it as you are grading it the first time; as this was a pair programming assignment, duplicates could still exist within the dataset. Participant 3 did not notice the duplicate and said during their post-study interview, ‘...like some assignments used the same animals... but [the implementation] was a little bit different so not like an exact copy of those two assignments...’.

In an effort to improve their consistency, 20 of the participants opted to review their grades. However, only 10 participants had a two or less grade difference between grading the first and second batches. While many opted to review their grades, many only gave a cursory review or reviewed if they thought they graded an assignment exceptionally high. Participant 12 stated, ‘...not too much time on all the of the like remarking all of them, just spending a minute on each assignment...’ and Participant 11 said ‘...if I graded someone high. I’ll go back and see if I graded anyone else at the same level...’. While the participants’ reviewing process may have increased their consistency for the grades they reviewed, it did not improve their overall consistency.

5.4.3 Grading Experience. Figure 11 shows the distribution by grading experience of the absolute distance between grades awarded in batch one and batch two for each of the four skills. Those with no prior grading experience have the largest difference between grades across all skills. Those with

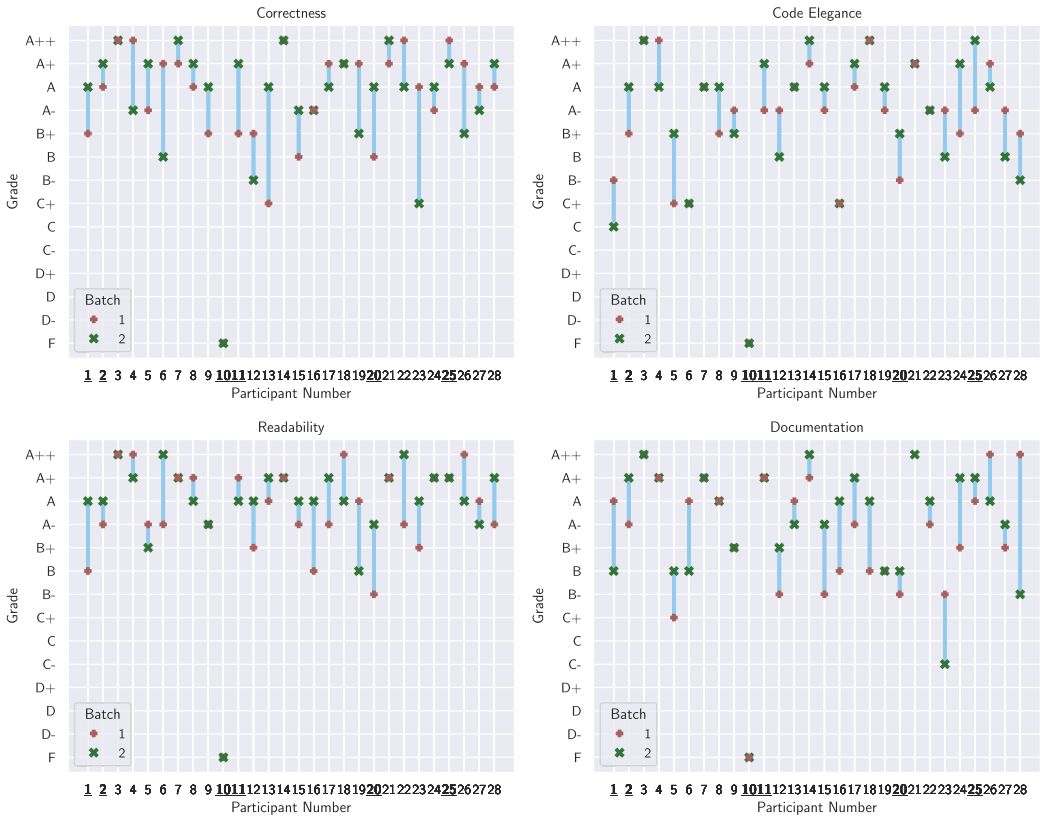


Fig. 10. The distance between the awarded grades for each skill in batch one and batch two for each participant when grading the duplicated assignment. The underlined participants indicated that they noticed the duplicate during the post-study interview. Participant 21 did not supply a grade for documentation in batch two, so it only shows one grade.

1–2 years of grading experience have at most a difference of three grades between the first and second time they graded the duplicated assignment, and those with one or more years of grading have a grade difference of less than one for most skills. However, readability and documentation for those with 2 or 3 years of grading experience had a maximum grade distance of two or three, respectively.

6 Discussion

6.1 How Consistently Does a Group of Graders Apply a Rubric? (RQ1)

Consistency was very poor overall. Even the best values achieved for Krippendorff's α reflect distinctly poor consistency, with none of our seven groups of graders achieving an $\alpha > 0.667$, the lower bound to consider reliable [35], with the average $\alpha = 0.105$ across all groups and skills.

While the correctness of a program is, in most cases, inherently objective, there can be elements of subjectiveness, especially within open-ended coursework assignments. The nature of open-ended assignments, where students decide how they implement a set of features using their ideas and creativity, can make it difficult for the groups of assessors to grade consistently. However, open-ended assignments allow instructors to design assignments that allow students to explore their creativity, which is a key educational goal [14, 52, 62], mimic professional development experiences, and improve students' self-efficacy in programming [57].

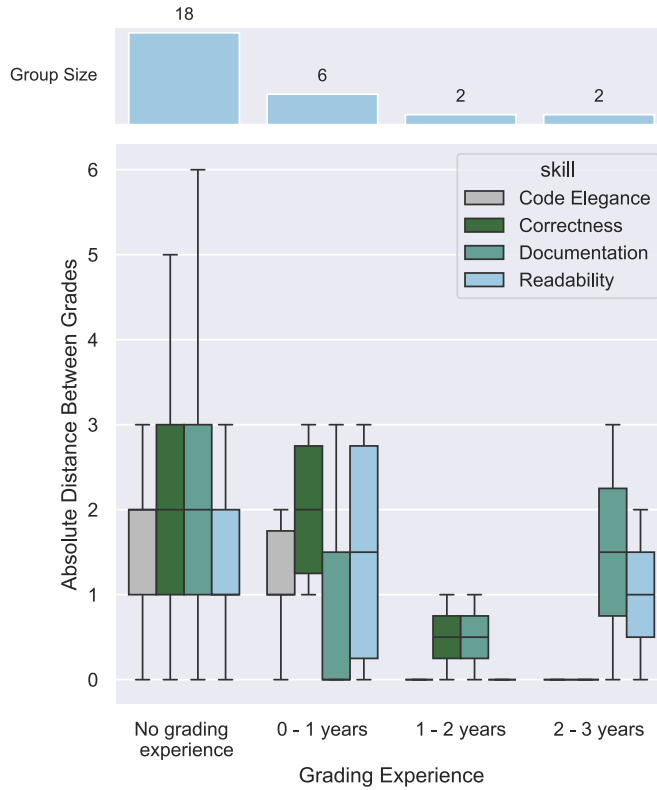


Fig. 11. This figure shows the distribution of the absolute distance between the awarded grades in batch one and batch two as a measure of the participant’s self-consistency for the participant’s grading experience and all four skills. The box plots’ box shows the inter-quartile range, while the whiskers extend to points within 1.5 of the inter-quartile range. The bar plot shows the proportion of participants with the varying levels of grading experience.

Furthermore, the subjectiveness of open-ended programming assignments extends not just to whether the code is correct and meets the requirements but to the quality of the code itself [32]. While some aspects of code quality are shared, such as consistent indentation and meaningful variable names, the nuances of programming style and design are individualised and remain active research topics not just within computer science education [33, 60], but within professional practice as well [39].

While having experienced graders may increase the consistency of assessment, this is not always possible as the typical demographics of teaching assistants are undergraduate, master’s or PhD students [44, 54, 55, 63], and many are new to assessment and have little experience; which is the case in our study. Having those with many years of grading experience is typically rare, as undergraduate teaching assistants are typically late-stage and are about to graduate, and graduate teaching assistants within Europe typically graduate within 3 to 4 years.

However, those with 1 year of grading experience had similar inter-rater reliability to those with no previous grading experience, indicating that more grading experience alone will not improve the grading consistency to an acceptable level. This supports the idea that to conduct a consistent assessment, assessors require adequate training to interpret and apply a rubric [36, 54], and that assessment is complex and intuitive [7, 28].

Rubric design plays an important role in the validity and consistency of the rubric [29]. The rubric provided to our participants follows the standard practice for rubric design [2, 51] and includes evaluation criteria, the definitions of quality and a scoring strategy.

However, as with the subjectiveness of programming, especially with open-ended assessment and code quality, as we have previously discussed, aspects of the rubric are subjective. Some aspects of the rubric are very clear, especially for the grades at either end of the grade spectrum. For example, for correctness, an F is defined as ‘Significant details of a task are violated, or the program often exhibits incorrect behaviour’. and an A+ – A++ is defined as ‘The application works as described in the assignment; all base tasks are completed; the implementation demonstrates originality, creativity and technical excellence...’.

Meanwhile, for the middle grades, the rubric becomes more subjective. For example, for correctness, a D is defined as ‘Some tasks are incomplete; the application functions incorrectly on some inputs/actions’. and a C is defined as ‘Minor details of the task(s) are violated; the application functions correctly on the majority inputs/actions’. Other examples from the rubric include D—code elegance, ‘The application is unnecessarily complex...’, D—documentation, ‘...the code is overly commented’, D—readability, ‘The application has more than two issues that make the program difficult to understand’. The lack of specificity in the rubric, especially for the middle grades, could have impacted the consistency of the grading, as different assessors can have differing opinions on what constitutes ‘some’ and ‘majority’ correct inputs and if the source code is unnecessarily complex, over-documented or difficult to understand.

There is likely to be less disagreement at either end of the grading spectrum, an F or an A+/A++, though rarely do any of these have complete agreement unless the submissions do not compile or do not contain any student code, with most groups being within one or two grades at these grades. Meanwhile, the middle grades are not graded consistently, with most groups having a grade difference of three or more.

While a more explicit rubric may improve the consistency of the grading, it could reduce the validity of the assignment [29], as a more explicit rubric would require a less open and creative assignment, limiting the students’ self-efficacy [57] and learning opportunities. Furthermore, even an explicit and analytic rubric may not infer consistent assessment, as grading consistently can be extremely difficult even with well-designed assessment criteria [28]. However, future work should investigate which aspects of code quality can be explicitly defined in the rubric and if defining these objectively improves consistency when grading open-ended assignments. Explicitly defined criteria could include explicitly defining code style conventions students should follow, such as naming conventions, avoiding single-character variable names, indentation, and spaces around operators.

6.2 Are Individual Graders Consistent at Applying a Rubric? (RQ2)

Compared to the consistency of multiple graders, readability, code elegance, and documentation achieved a higher individual consistency. The higher consistency of code elegance, readability, and documentation could be due to our participants being experienced programmers and having a well-defined internal preference for code elegance, readability and documentation [38, 45].

The lower consistency of correctness could be due to many of our participants having not graded before, which is typical when using teaching assistants [44, 54, 55, 63]. While some may have completed the assignment as a student, they would have only seen a limited set of other submissions. Another aspect could be the participants having to learn the specificity of what makes a submission good for this specific assignment; as the assignment is open-ended many different solutions can meet the requirements to the same standard, but with the specificity of the rubric, that standard is up interpretation by the grader. Similar to multiple grader consistency, a more specific rubric could increase the consistency, but at the cost of the validity [29] and creativity of the assignment.

Having more grading experience can lead to being more internally consistent when grading similar work, and supports the emphasis on providing appropriate training for teaching assistants [36, 63]. However, grading experience alone does not completely mitigate errors in consistency with experienced graders [54], especially when assessing subjective aspects like documentation and readability.

6.3 Implications for Human Grading

We have found that, in our study, typical graders are very inconsistent with each other and also are inconsistent with themselves. This suggests that current grading practices that use multiple assessors are likely to be quite unfair, which can lead to unfair outcomes for the students and demotivation among the students. Many suggest the importance of training assessors [8, 36, 63], and it may be that this is a necessary step to ensure fairness in grading.

We believe that our result warrants further consideration and investigation. Programming assignments are supposed to be relatively objectively judgeable and yet our research suggests that even with a shared rubric, inconsistency is the default outcome, which supports the idea that grading is ‘complex, intuitive and tacit’ [7]. This also lends support to the idea of investigating alternative grading approaches, such as those suggested by Decker et al. [17] for making grading more equitable. They propose changing the grading scales (such as making them coarser) in order to speed up grading and reduce disputes about exact grading scores—although our study already uses a coarser scale than 0–100 and found inconsistency even on a coarser scale.

6.4 Implications for Automated Grading

Part of our motivation for conducting this study was to get a ‘gold standard’ baseline human work in order to later investigate the accuracy of automated grading. However, our results suggest that the idea of a gold standard of human grading may be flawed, given that human graders cannot agree on the grade to give a piece of student work, even with themselves: inconsistency was high between and ‘within’ individual graders. This has strong implications for the acceptability of automated grading. One could argue that automated grading (including AI-based grading) needs to be within the same range as human grading to be considered acceptable, and since the range for human grading is wide, automated grading is more likely to be considered sufficient. However, this is an unconvincing argument. Rather than using the weakness of human grading as a convenient, low bar for assessing auto-graders, we—as a teaching community—should recognise and address the serious problem and try to find ways to improve the fundamental practice of grading student assignments. Automated grading may or may not have a part in this. One of the contributions of this article is a public release of our dataset, including the human grades, which can be used by all researchers for future work on automated grading.

7 Threats to Validity

7.1 Internal

One limitation of this study is that the participants knew it was a study that was not providing real grades and did not affect student outcomes. However, based on the interviews, many participants treated it as if they were marking for real, with many students opting to review their grades. The participants were aware that this was a study into grading consistency and, as such, may have altered their behaviour to maximise their internal consistency. While the participants may have altered their behaviour, our results still show significant inconsistency within their grades between a group of graders and when they graded an identical assignment later.

The rubric that was used in the original coursework provided generic descriptions for each grade, which may be less clear and well-defined than point-based rubrics that provide a detailed rubric and the points associated with each specific element. However, as the rubric was provided to the students when they originally submitted the coursework, we decided to mimic real-world assessment at our institution to provide an accurate insight into the consistency of assessment within this course.

We opted to provide only one identical submission in the second batch, which limits the validity of our evaluation of the internal consistency of human graders. We chose to only have one repeat to maximise the number of graded unique submissions, allowing for a more robust dataset when conducting future work. While only having one repeated assignment limits the validity of this part of our experiment, the results show that even with our small sample size, with each group of four graders grading a different duplicate, nearly all participants did not provide the same grade when grading the assignment a second time.

As we are using historical assignments from the CS1 programming course at our institution and many of our participants took that course during their undergraduate, there is a remote chance that the participants graded their own work. As these assignments were completely anonymised before undertaking this study, we could not tell if we asked the participants to mark their work. However, as the dataset of anonymised consists of over 600 submissions, the chances that one of the 272 graded submissions will be graded by its author is very low.

7.2 External

Our raw data and participants were all captured from one institution and one course, which limits how these results can be applied to other institutions or courses. Programming courses which use different rubrics, especially those that are more specific than the one provided in Appendix B, could have increased consistency. Prior work, such as by Passonneau et al. [49] and Jönsson et al. [28], suggests that simply making the rubric more detailed may not necessarily produce more consistent grading. Future work should evaluate how the specificity of the rubric affects consistency and the ability to provide open-ended assignments. Furthermore, different groups of graders, whether that be undergraduate, postgraduate teaching assistants, instructors, or a combination of all three, could result in varying grading consistency. Further work could investigate how different combinations of graders, training, and other methods can result in providing grades that are consistently applied between the group members.

As assessment is such a core aspect of education and students' prospects, we believe our results highlight a fundamental issue with how summative assessment is currently conducted, whether that is caused by instructors not providing detailed rubrics or utilising multiple teaching assistants with limited training in assessment. This fundamental issue is further supported by many institutions implementing rigorous assessment procedures to ensure that the assessment practices within their institutions are fair to all students, and indicates that rubrics should be seen as potentially useful addition to the range of assessment tools available and not a solution for all concerns raised by students in terms for quality, consistency and usefulness [15].

8 Conclusion

Inconsistent grading can impact students throughout their education and their post-education opportunities. Receiving inconsistent grades can confuse many students about why they received a particular grade, especially when comparing their grades with those of their peers, and affect their progression through their degree, especially if they do not meet course prerequisites, as well as affecting their sense of justice [47]. Post-education, inconsistent grading can affect a student's

career prospects, as many companies utilise degree classification or average grade in their hiring process [61].

Our study investigated the consistency of human grading by asking 28 participants to grade 272 authentic student assignments. We selected participants from our institution who were PhD or master's students with more than 3 years of programming knowledge; previous grading experience was not a requirement. This selection of participants mimics the demographics of the teaching assistants who complete the grading at many institutions, especially those with large cohorts, where the individual module/class leaders cannot feasibly grade the cohort in the required timeframe for feedback to be meaningful.

We split our 28 participants into groups of 4; each group graded the same 40 assignments in two batches of 20, with a two-week gap between them. To investigate the consistency of using multiple graders, we evaluated the inter-rater reliability using Krippendorff's α [35], which factors the ordinal nature of grades by applying weights to the metric calculation. We found that our participants provided inconsistent grades across correctness, code elegance, readability and documentation, with the most consistent being correctness with an average $\alpha = 0.2$ and code elegance, readability and documentation all having an average $\alpha < 0.1$ – where an $\alpha \geq 0.667$ is required to draw tentative conclusions and an $\alpha > 0.8$ suggests acceptable consistency [35]. This indicates a very high level of inconsistency of grading between graders.

In addition to investigating the consistency of multiple graders, we analysed the individual graders' self-consistency by duplicating one of the assignments in the first batch to the second batch with a different submission ID and measuring the difference between the grades they awarded in the first batch, and the grade they awarded in the second batch. Only one participant did not notice the duplicate assignment and graded consistently for all four skills. We found that an individual's self-consistency was, on average, higher for the subjective elements under assessment, code elegance, readability and documentation, compared to correctness, which was more inconsistent. The average grade difference was 1.79 for correctness, and the subjective elements all had an average grade difference of less than 1.6.

While these results are only from one assignment at one institution, which utilised a generic rubric, our study supports the notion that assessment is complex and intuitive [7] and that having a well-designed open-ended assignment may not imply consistent evaluation [29]. Our results suggest that there is a balance to be found between the openness of the assignment and the specificity of the rubric, and future research should investigate how aspects of code quality and correctness can be explicitly defined while allowing for an open-ended assignment. It also highlights that further research is required to improve the consistency of a group of graders and individual graders when grading open-ended programming assignments, whether it be alternative grading practices or methods of effectively training undergraduate and graduate teaching assistants.

8.1 Future Work

As assessment plays a crucial role in a student's education and future career prospects, we recommend that further research be undertaken on how to improve the consistency of grading programming assignments, whether that be investigating rubric design, the importance of assessor training, the use of alternative assessment approaches such as comparative judgement, or automated assessment tools.

In the future, we plan on expanding on this work by evaluating the consistency of the feedback provided as part of this study, including the topics on which the assessors gave feedback and the quality of their feedback.

Acknowledgements

We thank the King's College Teaching Fund for providing the funding necessary to undertake this research. We also thank Jeffery Raphael for providing the original data and Zara Lim and Nikolaj Jensen for anonymising the student assignments dataset. We thank Carlos Matos and Nuno Barreiro for their comments and ideas throughout the project and Mark Guzdial and Jarmoir Savelka for proofreading and commenting on the final draft. We thank the reviewers for their detailed and constructive reviews.

Data Availability

All our raw data and data analysis notebooks can be found on GitHub⁵. The Menagerie dataset, which includes all the anonymised student submissions and all the human grades gathered during this study, can be found on the Open Science Foundation⁶.

References

- [1] Tuukka Ahoniemi, Essi Lahtinen, and Tommi Reinikainen. 2008. Improving pedagogical feedback and objective grading. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*. ACM, New York, NY, 72–76. DOI: <https://doi.org/10.1145/1352135.1352162>.
- [2] Heidi Goodrich Andrade. 2005. Teaching with rubrics: The good, the bad, and the ugly. *College Teaching* 53, 1, (Jan. 2005), 27–31. DOI: <https://doi.org/10.3200/CTCH.53.1.27-31>
- [3] Marcus Messer, Neil Brown, Michael Kölling, and Miaojing Shi. 2024. How Consistent Are Human Graders? Pre-registration. Retrieved from https://osf.io/3nc7p/?view_only=d37833bc919d4a1f96d0fa0dcd43b5c7
- [4] Tapio Auvinen. 2011. Rubric. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11)*. ACM, New York, NY, 102–106. DOI: <https://doi.org/10.1145/2094131.2094152>
- [5] David Azcona and Alan Smeaton. 2020. +5 Million Python & Bash Programming Submissions for 5 Courses & Grades for Computer-Based Exams over 3 academic years. DOI: <https://doi.org/10.6084/m9.figshare.12610958.v1>
- [6] David John Barnes and Michael Kölling. 2006. *Objects First with Java: A Practical Introduction Using BlueJ*. Pearson/Prentice Hall, London, United Kingdom.
- [7] Sue Bloxham, Birgit den Outer, Jane Hudson, and Margaret Price. 2016. Let's stop the pretence of consistent marking: Exploring the multiple limitations of assessment criteria. *Assessment & Evaluation in Higher Education* 41, 3 (2016), 466–481. DOI: <https://doi.org/10.1080/02602938.2015.1024607>
- [8] Rodrigo Borela and Nimisha Roy. 2023. Creating equitable grading practices with rubrics: A teaching assistant training activity. In *Proceedings of the 2023 ACM Conference on International Computing Education Research (ICER '23)*. Association for Computing Machinery, New York, NY. Vol. 2, 26–27. DOI: <https://doi.org/10.1145/3568812.3603485>
- [9] Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Keuning Hieke, Kleiner Carsten, and Bonnie MacKellar. 2018. “I know it when I see it” perceptions of code quality: ITiCSE '17 working group report. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports (ITiCSE-WGR '17)*. ACM, New York, NY, 70–85. DOI: <https://doi.org/10.1145/3174781.3174785>
- [10] Carla E. Brodley. 2022. Why universities must resist GPA-based enrollment caps in the face of surging enrollments. *Communications of the ACM* 65, 8 (Jul. 2022), 20–22. DOI: <https://doi.org/10.1145/3544547>
- [11] Susan M. Brookhart. 2018. Appropriate criteria: Key to effective rubrics. *Frontiers in Education* 3 (2018), 22.
- [12] Neil C. C. Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. ACM, New York, NY, 223–228. DOI: <https://doi.org/10.1145/2538862.2538924>
- [13] Santiago Castro. 2017. Fast Krippendorff's alpha agreement measure. Retrieved from <https://github.com/pln-fing-udelar/fast-krippendorff>
- [14] Serene Chan and Mantak Yuen. 2014. Personal and environmental factors affecting teachers' creativity-fostering practices in Hong Kong. *Thinking Skills and Creativity* 12 (2014), 69–77. DOI: <https://doi.org/10.1016/j.tsc.2014.02.003>
- [15] Andrea Cockett and Carole Jackson. 2018. The use of assessment rubrics to enhance feedback in higher education: An integrative literature review. *Nurse Education Today* 69, (Oct. 2018), 8–13. DOI: <https://doi.org/10.1016/J.NEDT.2018.06.022>

⁵Data analysis repository: https://anonymous.4open.science/r/Consistency_In_Grading_Analysis-AC73/README.md.

⁶Menagerie: <https://osf.io/q8jbt/>.

- [16] Adrian de Freitas, Joel Coffman, Michelle de Freitas, Justin Wilson, and Troy Weingart. 2023. FalconCode: A multiyear dataset of python code samples from an introductory computer science course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*. ACM, New York, NY, 938–944. DOI : <https://doi.org/10.1145/3545945.3569822>
- [17] Adrienne Decker, Stephen H. Edwards, Brian M. McSkimming, Bob Edmison, Audrey Rorrer, and Manuel A. Pérez Quiñones. 2024. Transforming grading practices in the computing education community. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM, New York, NY, 276–282. DOI : <https://doi.org/10.1145/3626252.3630953>
- [18] Jacob Doughty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, Arav Agarwal, et al. 2024. *A comparative study of AI-generated (GPT-4) and human-crafted MCQs in programming education*. In *AWWA's Annual Conference & Exposition (ACE '24)*. ACM, New York, NY, USA, 114–123. DOI : <https://doi.org/10.1145/3636243.3636256>
- [19] Sabine S. S. Dunbar. 2018. Consistency in grading clinical skills. *Nurse Education in Practice* 31 (2018), 136–142.
- [20] Bobbie Lynn Eicher and David Joyner. 2021. Components of assessments and grading at scale. In *Proceedings of the 8th ACM Conference on Learning @ Scale (L@S '21)*. ACM, New York, NY, 303–306. DOI : <https://doi.org/10.1145/3430895.3460165>
- [21] Kathi Fisler. 2014. The recurring rainfall problem. In *Proceedings of the 10th Annual Conference on International Computing Education Research*. ACM, New York, NY, 35–42. DOI : <https://doi.org/10.1145/2632320.2632346>
- [22] Fay Fransella, Richard Bell, and Don Bannister. 2004. *A Manual for Repertory Grid Technique*. John Wiley & Sons.
- [23] Alex Gerdes, Johan T. Jeuring, and Bastiaan J. Heeren. 2010. Using strategies for assessment of programming exercises. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*. ACM, New York, NY, 441–445. DOI : <https://doi.org/10.1145/1734263.1734412>
- [24] Nirit Glazer. 2014. Formative plus summative assessment in large undergraduate courses: Why both? *International Journal of Teaching and Learning in Higher Education* 26, 2 (2014), 276–286.
- [25] Kilem Li Gwet. 2008. Computing inter-rater reliability and its variance in the presence of high agreement. *The British Journal of Mathematical and Statistical Psychology* 61, 1 (2008), 29–48. DOI : <https://doi.org/10.1348/000711006X126600>
- [26] Charles Henderson, Edit Yerushalmi, Vince H. Kuo, Patricia Heller, and Kenneth Heller. 2004. Grading student problem solutions: The challenge of sending a consistent message. *American Journal of Physics* 72, 2 (Feb. 2004), 164–169. DOI : <https://doi.org/10.1119/1.1634963>
- [27] Nathan M. Hicks and Heidi A. Diefes-Dux. 2017. Grader consistency in using standards-based rubrics. In *Proceedings of the 2017 ASEE Annual Conference & Exposition*.
- [28] Anders Jönsson, Andreia Balan, and Eva Hartell. 2021. Analytic or holistic? A study about how to increase the agreement in teachers' grading. *Assessment in Education: Principles, Policy & Practice* 28, 3 (2021), 212–227. DOI : <https://doi.org/10.1080/0969594X.2021.1884041>
- [29] Anders Jönsson and Gunilla Svingby. 2007. The use of scoring rubrics: Reliability, validity and educational consequences. *Educational Research Review* 2, 2 (2007), 130–144. DOI : <https://doi.org/10.1016/j.edurev.2007.05.002>
- [30] David Kane, James Williams, and Gillian Cappuccini-Ansfield. 2008. Student Satisfaction Surveys: The Value in Taking an Historical Perspective. 2 (Jul. 2008), 135–155. DOI : <https://doi.org/10.1080/13538320802278347>
- [31] Jennifer S. Kay. 2022. Peer grading without protest: The SPARK approach to summative peer assessment. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education (SIGCSE '22)*. ACM, New York, NY, 119–125. DOI : <https://doi.org/10.1145/3478431.3499284>
- [32] Diana Kirk, Andrew Luxton-Reilly, and Ewan Tempero. 2024. Code style != code quality. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1 (SIGCSE Virtual '24)*. ACM, New York, NY, 267–270. DOI : <https://doi.org/10.1145/3649165.3703621>
- [33] Diana Kirk, Andrew Luxton-Reilly, and Ewan Tempero. 2024. A literature-informed model for code style principles to support teachers of text-based programming. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*. ACM, New York, NY, 134–143. DOI : <https://doi.org/10.1145/3636243.3636258>
- [34] Peter T. Knight. 2002. Summative assessment in higher education: Practices in disarray. *Studies in Higher Education* 27, 3 (2002), 275–286.
- [35] Klaus Krippendorff. 2013. *Content Analysis: An Introduction to Its Methodology* (3rd ed.). Sage Publications, Los Angeles.
- [36] Nynne Grauslund Kristiansen, Sebastian Mateos Nicolajsen, and Claus Brabrand. 2024. Feedback on student programming assignments: Teaching assistants vs automated assessment tool. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23)*. ACM, New York, NY. DOI : <https://doi.org/10.1145/3631802.3631804>
- [37] Kathleen J. Lehman, Julia Rose Karpicz, Veronika Rozhenkova, Jamelia Harris, and Tomoko M. Nakajima. 2021. Growing enrollments require us to do more: Perspectives on broadening participation during an undergraduate

- computing enrollment boom. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. ACM, New York, NY, 809–815. DOI: <https://doi.org/10.1145/3408877.3432370>
- [38] Xuechao Li, Po-Chou Shih, and Evans David. 2018. The effect of software programmers' personality on programming performance. In *Proceedings of the 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 209–213. DOI: <https://doi.org/10.1109/ICAIBD.2018.8396196>
- [39] Jenny T. Liang, Maryam Arab, Minhyuk Ko, Amy J. Ko, and Thomas D. LaToza. 2023. A qualitative study on the implementation design decisions of developers. In *Proceedings of the 45th International Conference on Software Engineering (ICSE '23)*. IEEE Press, 435–447. DOI: <https://doi.org/10.1109/ICSE48619.2023.00047>
- [40] Soohyun Nam Liao, William G. Griswold, and Leo Porter. 2017. Impact of class size on student evaluations for traditional and peer instruction classrooms. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, 375–380. DOI: <https://doi.org/10.1145/3017680.3017764>
- [41] Ian C. McManus, Margaret Thompson, and Jennifer Mollon. 2006. Assessment of examiner leniency and stringency ('hawk-dove effect') in the MRCP(UK) clinical examination (PACES) using multi-facet rasch modelling. *BMC Medical Education* 6, 1 (2006), 42. DOI: <https://doi.org/10.1186/1472-6920-6-42>
- [42] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaoqing Shi. 2024. Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computer Education* 24, 1, Article 10 (Feb. 2024), 43 pages. DOI: <https://doi.org/10.1145/3636515>
- [43] Gosia Migut and Ruben Wiersma. 2020. Are we consistent? The effects of digitized exams grading. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. ACM, New York, NY, 1338. DOI: <https://doi.org/10.1145/3328778.3372630>
- [44] Diba Mirza, Phillip T. Conrad, Christian Lloyd, Ziad Matni, and Arthur Gatin. 2019. Undergraduate teaching assistants in computer science: A systematic literature review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. ACM, New York, NY, 31–40. DOI: <https://doi.org/10.1145/3291279.3339422>
- [45] Andrew Mohan and Nicolas Gold. 2004. Programming style changes in evolving source code. In *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, 236–240. DOI: <https://doi.org/10.1109/WPC.2004.1311066>
- [46] Aida Mustapha, Noor Azah Samsudin, Nurieze Arbaiy, Rozlini Mohammed, and Isredza Rahmi Hamid. 2016. Generic assessment rubrics for computer programming courses. *Turkish Online Journal of Educational Technology - TOJET* 15, 1 (Jan. 2016), 53–68.
- [47] Paul L. Nesbit and Suzan Burton. 2006. Student justice perceptions following assignment feedback. *Assessment & Evaluation in Higher Education* 31, 6 (2006), 655–670. DOI: <https://doi.org/10.1080/02602930600760868>
- [48] Tom Neutens, Kris Coolsaet, and Francis Wyffels. 2022. Assessment of code, which aspects do teachers consider and how are they valued? *ACM Transactions on Computer Education* 22, 4, Article 46 (Sep. 2022), 27 pages. DOI: <https://doi.org/10.1145/3517133>
- [49] Rebecca J. Passonneau, Kathleen Koenig, Zhaohui Li, and Josephine Soddano. 2023. The ideal versus the real deal in assessment of physics lab report writing. *European Journal of Applied Sciences* 11, 2 (Apr. 2023), 626–644. DOI: <https://doi.org/10.14738/aivp.112.14406>
- [50] Sawan Rai, Ramesh Chandra Belwal, and Atul Gupta. 2022. A review on source code documentation. *ACM Transactions on Intelligent System Technology* 13, 5, Article 84 (Jun. 2022), 44 pages. DOI: <https://doi.org/10.1145/3519312>
- [51] Y. Malini Reddy and Heidi Andrade. 2010. A review of rubric use in higher education. *Assessment & Evaluation in Higher Education* 35, 4 (2010), 435–448. DOI: <https://doi.org/10.1080/02602930902862859>
- [52] Carmen Richardson and Punya Mishra. 2018. Learning environments that support student creativity: Developing the SCALE. *Thinking Skills and Creativity* 27 (2018), 45–54. DOI: <https://doi.org/10.1016/j.tsc.2017.11.004>
- [53] Emma Riese. 2018. Teaching assistants' experiences of lab sessions in introductory computer science courses. In *Proceedings of the 2018 IEEE Frontiers in Education Conference (FIE)*, 1–5. DOI: <https://doi.org/10.1109/FIE.2018.8659243>
- [54] Emma Riese and Olle Bälter. 2022. A qualitative study of experienced course coordinators' perspectives on assessment in introductory programming courses for non-CS majors. *ACM Transactions on Computer Education* 22, 4, Article 45 (Sep. 2022), 29 pages. DOI: <https://doi.org/10.1145/3517134>
- [55] Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 2021. Challenges faced by teaching assistants in computer science education across Europe. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE '21)*. ACM, New York, NY, 547–553. DOI: <https://doi.org/10.1145/3430665.3456304>
- [56] Ilona Rinne. 2024. Same grade for different reasons, different grades for the same reason? *Assessment & Evaluation in Higher Education* 49, 2 (2024), 220–232. DOI: <https://doi.org/10.1080/02602938.2023.2203883>
- [57] Sadia Sharmin, Daniel Zingaro, Lisa Zhang, and Clare Brett. 2019. Impact of open-ended assignments on student self-efficacy in CS1. In *Proceedings of the ACM Conference on Global Computing Education (CompEd '19)*. ACM, New York, NY, 215–221. DOI: <https://doi.org/10.1145/3300115.3309532>

- [58] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. 2017. Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work. In *Proceedings of the 4th (2017) ACM Conference on Learning @ Scale (L@S '17)*. ACM, New York, NY, 81–88. DOI: <https://doi.org/10.1145/3051457.3051466>
- [59] Elliot Soloway. 1986. Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM* 29, 9 (1986), 850–858. DOI: <https://doi.org/10.1145/6592.6594>
- [60] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2016. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16)*. ACM, New York, NY, 160–164. DOI: <https://doi.org/10.1145/2999541.2999555>
- [61] Anna Stepanova, Alexis Weaver, Joanna Lahey, Gerianne Alexander, and Tracy Hammond. 2021. Hiring CS graduates: What we learned from employers. *ACM Transactions on Computer Education* 22, 1, Article 5 (Oct. 2021), 20 pages. DOI: <https://doi.org/10.1145/3474623>
- [62] Tony Wagner. 2010. *The Global Achievement Gap: Why Even Our Best Schools Don't Teach the New Survival Skills Our Children Need-and What We Can Do about It*. Retrieved from ReadHowYouWant.com
- [63] Navé Wald and Tony Harland. 2018. Rethinking the teaching roles and assessment responsibilities of student teaching assistants. *Journal of Further and Higher Education* 44, 1 (2018), 43–53. DOI: <https://doi.org/10.1080/0309877X.2018.1499883>
- [64] Keith Willey and Anne Gardner. 2010. Perceived differences in tutor grading in large classes: Fact or fiction? In *Proceedings of the 2010 IEEE Frontiers in Education Conference (FIE)*, S2G–1–S2G–6. DOI: <https://doi.org/10.1109/FIE.2010.5673351>
- [65] Nahathai Wongpakaran, Tinakon Wongpakaran, Danny Wedding, and Kilem L. Gwet. 2013. A comparison of Cohen's kappa and Gwet's AC1 when calculating inter-rater reliability coefficients: A study conducted with personality disorder samples. *BMC Medical Research Methodology* 13, 1 (2013), 61. DOI: <https://doi.org/10.1186/1471-2288-13-61>
- [66] Benjamin D. Wright. 1977. Solving measurement problems with the Rasch model. *Journal of Educational Measurement* 14, 2 (1977), 97–116.

Appendices

A Further Details on the Menagerie Dataset

A.1 Data Processing

This project was a small-group project, with students in groups of two (or rarely three). Each student was required to submit a copy of the assignment, so all submissions were duplicated at least once. We first generated hashes to remove these duplicated assignments.

Additionally, we manually reviewed potential duplicates based on the number of classes and the number of source lines of code; if they were identical, the copy was removed. After removing the duplicates, we extracted the students' submissions and removed unnecessary files, such as IDE property files and .class files.

To anonymise the dataset, we automatically removed the JavaDoc @author tag lines in all classes. The @author tag is typically used to identify who wrote the class and would be the most likely place for a student's name or identification number to appear in the code. To verify that no further personally identifiable information was left in the source code and to minimise duplicated submissions, we generated a list of changes made by the student compared to the template code. Multiple research assistants then used the list of changes to review the submitted code and remove any remaining personally identifiable information. Our data anonymisation approach limits the potential for students to be identified from the anonymised submission while retaining all other information.

During the anonymisation process, we found that many projects opted to include image files as part of their project. As these cannot be removed as they would interfere with how the submission is executed and cannot be anonymised easily, we opted to replace all images with equivalently sized black images.

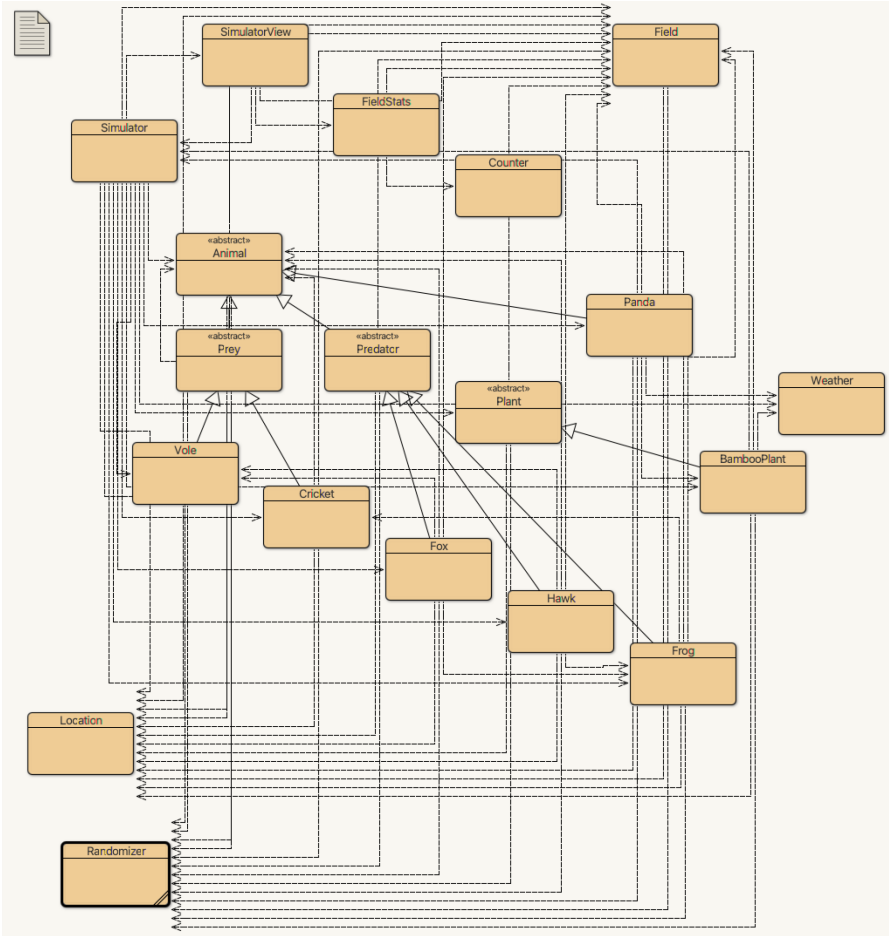


Fig. A1. The UML Diagram of the median-sized submission by class size and source lines of code.

A.2 An In-Depth Example Submission

To give a specific example of an assessment submission, we picked submission 189, which is median-sized regarding the number of classes implemented and the number of code source lines, with 19 classes and 1,345 source lines of code. Figure A1 is the final UML of the submission and, by comparison to Figure 1, shows that the students significantly expanded on the template project.

To complete the tasks, as defined in Section 3.1.1, the students implemented the five species, with at least two being prey and at least two being predators, using inheritance and abstract classes to implement shared functionality. All of the five species classes inherit from the Animal class, with the prey and predators inheriting the Animal class indirectly through their parent classes, Prey and Predator respectively (which were not present in the original code, and have been added as part of the object-oriented design by the students). Both the Fox and Hawk species compete for Voles and Frogs, and Frogs hunt for Crickets by implementing the abstract method hunt from the Predator class and using instanceof to validate if the adjacent locations contain one of the prey.

To distinguish between male and female animals, the students added a Boolean field and a function to randomly assign male or female in the Animal class. In each species class, the students

added functions to validate whether the two members of the same species were adjacent and of opposite genders. If so, they add a new animal to a space in the two-dimensional array.

To incorporate the time of day into their submission, the students added fields `hour`, `day` and `dayNightStatus` to the `Simulator` class, and a `isDay` field to the `Field` class. The `Cricket` is the only species with different actions during the day or night. During the day and night, the `Cricket`'s age increases, and they handle the disease, as explained later in this section, and at night they also propagate.

For the challenge tasks, the students implemented all three suggestions outlined in Section 3.1.1. To simulate the lifecycle of plants, the students added an abstract `Plant` class and concrete class `BambooPlant`. The plant propagated based on a probability to produce seeds and was food for the `Panda` class, which inherited directly from the `Animal` class.

The students implemented the four seasons to implement weather, which changed every 90 time steps. The `BambooPlant` has different functionality depending on the season. In the spring, it gets water, grows, and spreads, whereas in the winter, it loses water and does not grow or spread.

Finally, the students implemented a disease that affects all animals by adding an `isInfected`, probability of infection and death fields, and functions to catch, spread and check if the animal is infected to the `Animal` class. These functions were called as part of each animal's act methods, which were responsible for simulating each object at each timestep.

B Rubric

The rubric below was used in our study. It was provided to the students when they originally wrote their submissions.

B.1 Grade A+–A++ (80–100)

Program Correctness. The application works as described in the assignment; all base tasks are completed; the implementation demonstrates originality, creativity and technical excellence in the completion of all challenging tasks the student included documentation describing the challenge tasks that were completed and how; all submission instructions were followed correctly.

Code Elegance. Student demonstrates excellent use of classes and functions to produce reusable and maintainable code, where possible, in the base and all of the challenge tasks; the code is efficient without sacrificing readability and understanding.

Documentation. The documentation is well written, organised and clearly explains what the code is accomplishing and how; the student documents the contents of each class at the beginning of the file; the student documents the purpose of each function, i.e., the function's parameters and return values at the beginning of each function; the student documents each logical block of code when performing non-obvious operations; Every file includes header information (e.g., student name and id).

Readability. The application is exceptionally well organised and very easy to understand; the student used indentation appropriately and consistently to delineate code blocks; each function performs a single well defined operation; the student used meaningful identifier names, i.e., good function and variables names; the student used white space between logical code blocks; the student uses consistent spacing around operators and variables; classes are self contained with private data hidden and methods are public only when necessary.

B.2 Grade A (70–79)

Program Correctness. The application works as described in the assignment; all base tasks are completed; the student has completed some of the challenge tasks; the student included documentation describing the challenge tasks that were completed and how; all submission instructions were followed correctly.

Code Elegance. All of the base tasks and some of the challenge tasks are implemented in such a way that code can be reused, where possible; the majority of the code is written in such a way that is easy to maintain, i.e., add new or extend features; the code is efficient without sacrificing readability and understanding.

Documentation. The documentation consists of comments that are useful in understanding the code and/or structure of the program; each file has header information (e.g., student name and id).

Readability. The code is clean, understandable and organised; the student used meaningful identifier names, i.e., good function and variables names: the student used white space between logical code blocks.

B.3 Grade B (60–69)

Program Correctness. The application works as described in the assignment; all the base tasks are completed, however, no challenge task was attempted: all submission instructions were followed correctly.

Code Elegance. The application is implemented in such a way that only a few code segments could be rewritten to increase code re usability; the code is fairly efficient without sacrificing readability and understanding; the student made a poor design choice in a single area.

Documentation. The documentation consists of comments that are somewhat useful in understanding the code; some of the comments state the obvious.

Readability. The application has minor issues such as inconsistent indentation.

B.4 Grade C (50–59)

Program Correctness. Minor details of a task(s) are violated; the application functions correctly on the majority inputs/actions.

Code Elegance. The application is implemented in such a way that many code segments (or functions) could be rewritten to increase code re-usability: some of the code is unnecessarily complex or poorly designed.

Documentation. The documentation is simply comments embedded in the code but does not help the reader understand the code.

Readability. The application has one or two issues that makes the program difficult to understand such as poorly named identifiers and disorganised code.

B.5 Grade D (40–49)

Program Correctness. Some tasks are incomplete; the application functions incorrectly on some inputs/ actions.

Code Elegance. The application is unnecessarily complex and/or uses brute force; the application contains many instances where the code could have been written in an easier, faster or better fashion.

Documentation. One or more code segments could benefit from comments or the code is overly commented.

Readability. The application has more than two issues that makes the program difficult to understand.

B.6 Grade F (0–39)

Program Correctness. Significant details of a task are violated, or the program often exhibits incorrect behaviour; the applications does not open or run in `RueJ`; Java files are missing

Code Elegance. The application is not organised for re-usability, e.g., all the code is in a single class (or function); no effort is made to create reusable code; the application is excessively long and poorly organised.

Documentation. The application has no comments, not even the header with the student's name or id.

Readability. The code is readable only by someone who knows what it is supposed to do; the application has several issues that makes the program difficult to understand.

C Grades Per Assignment

This section shows all our results for the minimum, maximum and mean grade per group and per skill:

- Figure [C1](#) shows the minimum, maximum and mean grades for each of the seven groups when grading correctness.
- The mean, maximum, and minimum grades for each of the seven groups when grading code elegance are displayed in Figure [C2](#).
- Figure [C3](#) shows the minimum, maximum and mean grades for each of the seven groups when grading readability.
- The mean, maximum, and minimum grades for each of the seven groups when grading documentation are displayed in Figure [C4](#).

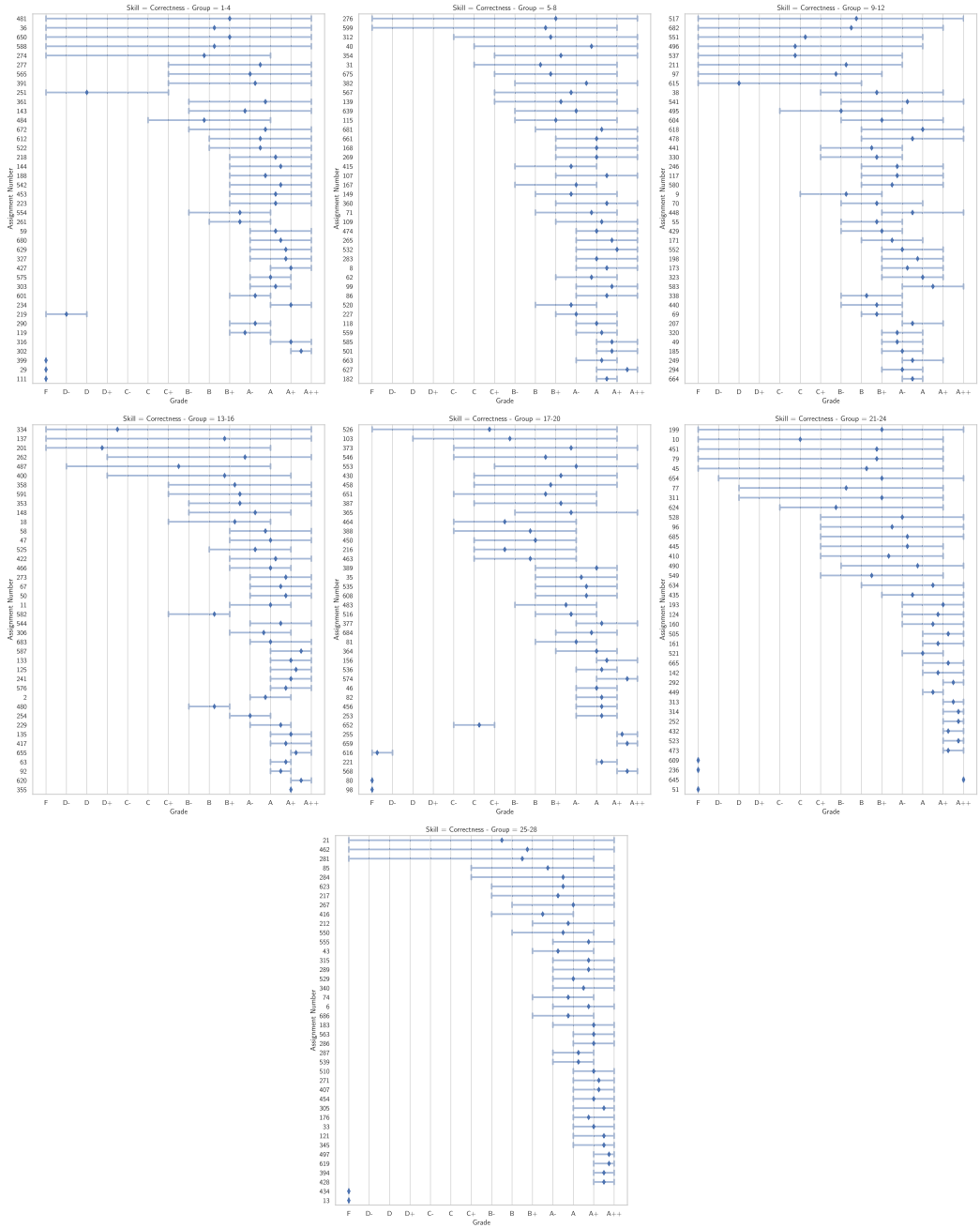


Fig. C1. These figures show the minimum, maximum and mean correctness grade awarded by the participants for each assignment.



Fig. C2. These figures show the minimum, maximum and mean code elegance grade awarded by the participants for each assignment.

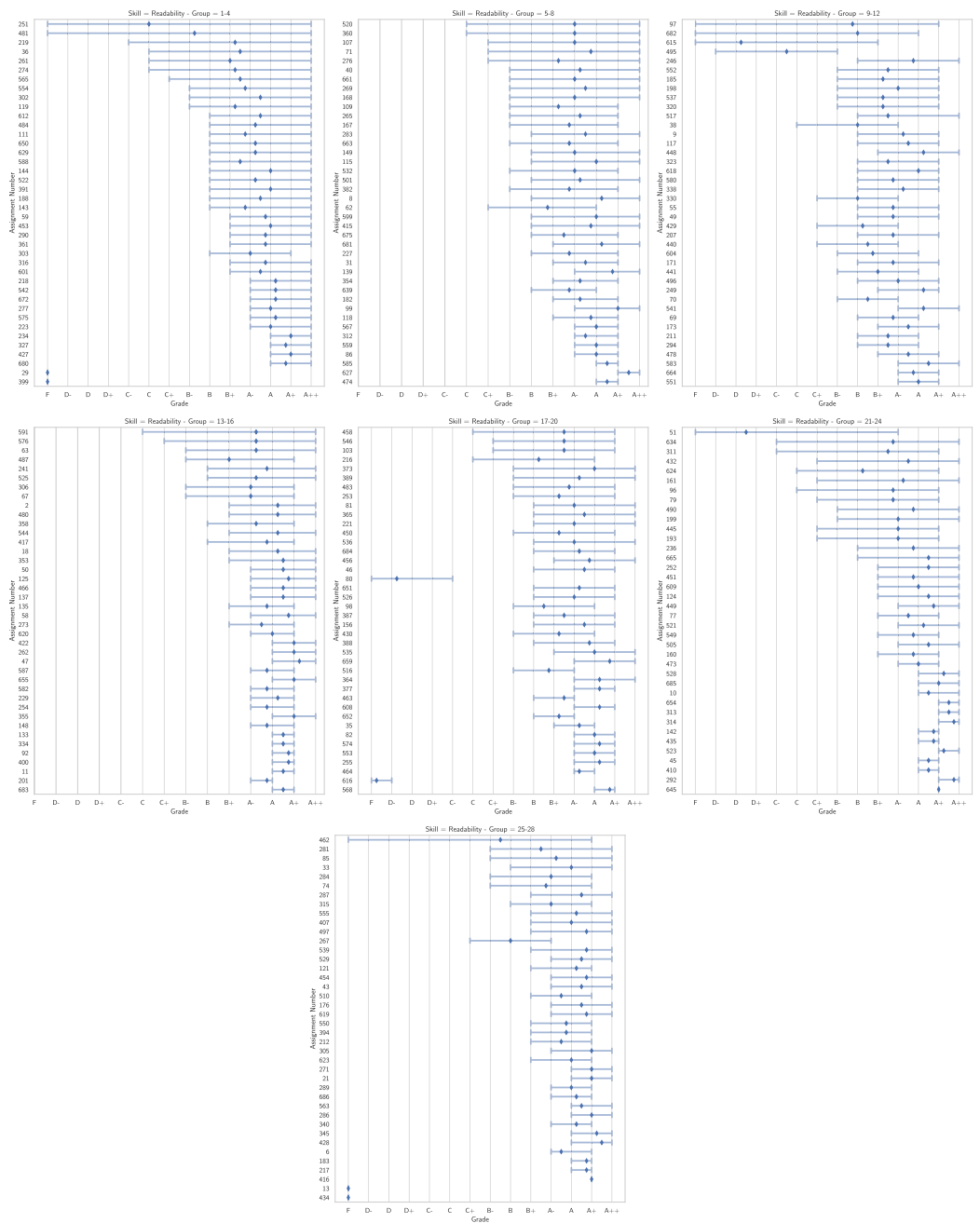


Fig. C3. These figures show the minimum, maximum and mean readability grade awarded by the participants for each assignment.



Fig. C4. These figures show the minimum, maximum and mean documentation grade awarded by the participants for each assignment.

Received 29 August 2024; revised 23 July 2025; accepted 31 July 2025