The Future of Engineering Education
2024 Annual Conference & Exposition
Oregon Convention Center
Portland, OR . June 23 - 26, 2024
ASEE
Paper ID #41987

Analyzing Individual Contribution in Team-based Software Engineering Projects

Joydeep Mitra, Northeastern University

Joydeep Mitra is an assistant teaching professor in the Khoury College of Computer Sciences at Northeastern University, based in Boston.

Mitra's research interests center around mobile app security and privacy. He has been awarded two Android security rewards by Google for discovering vulnerabilities in the Android platform. Further, he is passionate about the opportunity to train the next generation of computing professionals. He gets immense joy from teaching students how their favorite programming languages work under the hood, as well as developing new teaching methods and evaluating existing ones to understand what engages students. He was previously awarded a Faculty Fellowship by Stony Brook University to study the effects of the Process Oriented Guided Inquiry Learning methodology in a large classroom.

Mitra has published in the Journal of Empirical Software Engineering and the Technical Symposium of Computer Science Education (SIGCSE TS), and has presented at both the International Conference on Predictive Models and Data Analytics in Software Engineering and the International Workshop on Advances in Mobile App Analysis. Additionally, he has served as a journal reviewer for SIGCSE TS and Transactions in Software Engineering and Methodology.

Amir Kirsh

Measuring Individual Contribution in Team-based Software Engineering Projects

Joydeep Mitra
j.mitra@northeastern.edu
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA

Amir Kirsh
kirshamir@gmail.com
The Academic College of Tel Aviv Yaffo
Tel Aviv, Israel

Introduction

Collaboration in Computer Science courses has several benefits. It allows students with diverse backgrounds and perspectives to come together and understand the subject material holistically and comprehensively. Working in a team encourages students to exchange ideas, expertise, and best practices, which helps them learn from one another and not only from the teaching staff. When students work on programming assignments in a team, it allows students to peer review their team members' code. Peer evaluation and feedback help improve the work's overall quality and also reflects developers' workflow in real-world software development projects. Moreover, a collaborative environment encourages students to be accountable for their and their team members' work, enabling them to be disciplined and responsible.

While collaboration has numerous benefits, one of its primary challenges is for instructors to evaluate individual contributions in group or team projects. Teamwork in programming assignments may lead to unequal work distribution. Some students may do no or less work than others for various reasons ranging from lack of effective time management and insufficient background/skills to apathy and negligence. If students are not graded, or at least perceive that they can be graded based on individual work, some may be incentivized to work less than their teammates. Moreover, unequal work distribution, if undetected, may create an environment of resentment. Also, students who fail to do the necessary work may proceed without satisfying the desired learning objectives.

One potential method to ensure equal work distribution in group programming assignments is to have each team member work on individual source control branches, having them know that the course staff will verify their individual contributions during evaluation.

Given the benefits of collaborative work and the need to assess individual contributions, in this paper, we focus on answering the following research questions:

1. **RQ1(a)**: Do students in an introductory software engineering course, working in pairs, distribute work equally?
   RQ1(b): Can we use measurements based on git logs to assess each team member's work contribution? How is this measurement correlated with the amount of work self-reported by students?
2. **RQ2**: Would there be any change in their work distribution if given a recommended collaboration workflow?
3. **RQ3**: What collaborative models do students in an introductory software engineering course follow when working in a team?

Related Work

Working in pairs on programming assignments is helpful for students as it improves performance and self-efficacy [1, 2, 3, 4, 5]. However, assessing individual work in group activities is challenging. Therefore, several approaches to effectively evaluate group work have been proposed – (a) give the same grade to all students; (b) give the same grade to all students unless otherwise requested by the team or based on the instructor's perception that the work was unequal; (c) differentiate between students according to their individual contribution.

There are subjective and objective ways to determine individual contributions in a team. Subjective measures include peer evaluation and oral interviews organized by the course staff [6, 7, 8]. As subjective measures are subject to bias, researchers have also proposed considering objective measures based on version control systems such as git logs for a complete picture of individual contribution [9, 10, 11]. However, recent efforts have found that objective and subjective measures of individual contribution may not be correlated [12, 13]. Moreover, objective metrics can be narrow. For example, estimates derived from git logs may not capture individual contributions comprehensively as git logs only consider code contributions and not other forms (e.g., design) [14, 15].

Our effort contributes to the existing discussion on identifying metrics to accurately measure individual contributions in collaborative programming projects. We use a combination of objective (based on git logs) and subjective (self-reported effort) measures to determine individual contributions. Furthermore, unlike prior efforts, we also analyze and report the collaborative practices of students working in teams.

Methodology

Logistics. Our study is based on an undergraduate course on software development fundamentals taught in Spring 2023 at a large public research university in the United States. Students in the class paired up in teams and developed a web application mimicking the Stack Overflow Q&A forum [16]. Students created the application over the course of three programming assignments and a final project. In the first assignment, teams used HTML/CSS/JavaScript to build the user interface of the fake Stack Overflow application. In the second assignment, teams used the React framework to re-create the front-end features. In the third assignment, teams added a back-end server (Node.js) and data persistence using a non-relational database (MongoDB). In the final project, teams extended the application by adding advanced features (e.g., authentication).

We used GitHub Classroom [17] to administer the programming assignments and the final project. GitHub Classroom allows instructors to create a repository for each team. Students collaborated in their teams using the git repositories. Team members had read and write access to the repositories, meaning they could commit changes to the repository unlimited times until a deadline, create, merge, and close branches, and create and submit pull requests.

We restricted the team size to two. Students were allowed to form their teams. However, they were not permitted to change once they formed a team. Students who were unable to form teams were allowed to work individually. We do not consider students who worked alone in our analysis.

Overall, 110 students enrolled in the course. They formed 48 teams. The remaining 14 were allowed to work alone as they could not create teams. 3 of the 48 teams were not considered in the evaluation as one of their members dropped the course.

Experimental Design, Instruments, and Metrics. For each assignment and the final project, we analyzed the repositories of all teams in GitHub classroom. For each repository, we analyzed the commit history of the main branch. The commit history records the number of lines a team member added and deleted in each commit. We ignored merges since merges are not novel code contributions. Also, we ignored lines deleted to avoid double counting of lines added. We considered the total lines of code added by a team member across all commits as their contribution and calculated their ratio. We call this ratio the *work ratio*. The work ratio should be 1 (or close to 1) for equal contribution. Consider an example to understand the idea of work ratio better. Suppose in a team of two members – Alice and Bob, Alice added N1 lines of code, and Bob added N2 lines of code across several commits. Assuming $N1 < N2$, the work ratio N1/N2 indicates that Alice contributed N1/N2 of Bob's lines of code. If Alice and Bob contributed equally, the work ratio would be 1.

Furthermore, we calculated a *total work ratio* – the ratio of the total lines of code added by each team member across all assignments and the final project. For example, if Alice added A1, A2, A3, and AP1 lines of code in assignments 1-3 and the final project, respectively, and Bob added B1, B2, B3, and BP1 lines of code in assignments 1-3 and the final project, then the *total work ratio* is (A1+A2+A3+AP1)/(B1+B2+B3+BP1). This metric is necessary because it is possible

	S_X/P_X	S_Y/P_Y	P_X/P_Y	S_X/S_Y
S_X/P_X	NA	0.27 / 0.27 /0.44*	0.09 / 0.03 / 0.38*	0.44* / 0.16 / 0.36*
S_Y/P_Y	0.27 / 0.27 /0.44*	NA	0.64* / 0.18 / 0.5*	0.53* / 0.31* / 0.46*
P_X/P_Y	0.09 / 0.03 / 0.38*	0.64* / 0.18 / 0.5*	NA	**0.68* / 0.71* / 0.65***
S_X/S_Y	0.44* / 0.16 / 0.36*	0.53* / 0.31* / 0.46*	**0.68* / 0.71* / 0.65***	NA

Table 1: Comparison of time-based ratios based on Pearson Correlation Coefficient for Assignments 1-2 and final project. Each cell has a notation I/J/K, where I, J, and K are correlation coefficients of the compared ratios for assignments 1, 2, and the final project, respectively. The cell highlighted in bold demonstrates the strongest correlation. The coefficients with * indicate that they are statistically significant, assuming $\alpha = 0.05$ and a two-tailed t-test.

that team members negotiated with each other and split the work such that one works more in one assignment and the other balances it out by working more in the subsequent assignment/s. If the contribution between the team members is equal across all assignments, then the total work ratio should be close to 1.

Additionally, we used the time taken by each team member to complete the assignments as a metric to determine if team members distributed work equally. Hence, at the end of each assignment (except assignment 3), we ran a survey to collect student responses[1]. For each team comprising members X and Y, we requested the following:

1. S_X: the number of hours X worked individually as reported by X
2. P_X: the number of hours partner Y worked individually as estimated by X.
3. S_Y: the number of hours Y worked individually as reported by Y
4. P_Y: the number of hours partner X worked individually as estimated by Y.

We calculated four ratios – S_X/P_X, S_Y/P_Y, P_X/P_Y, and S_X/S_Y for each team based on the response given by its members X and Y The ratio S_X/P_X is an estimate of the work distribution as reported by X and S_Y/P_Y is the same estimate as reported by Y. The ratio P_X/P_Y indicates the work distribution of a team based on hours worked as reported for a team member by their partner. On the other hand, the ratio S_X/S_Y is the work distribution based on hours worked as reported by the team member themselves. For each metric, the ratio 1 indicates that each member contributed equally in terms of hours worked. However, we did not consider all four ratios in our evaluation, as some are more trustworthy than others. The ratios S_X/P_X and S_Y/P_Y can inaccurately represent work distribution as a team member may incorrectly estimate their partner's hours to be the same as theirs. On the other hand, the ratios P_X/P_Y and S_X/S_Y more accurately represent work distribution even if a student misreports their partner's hours to be the same as theirs. Additionally, we compared each ratio pair to determine if they are correlated (see Table 1). A strong correlation between any pair of ratios indicates that ratios in the pair agree with each other

[1]We could not collect the responses at the end of the third assignment due to time conflicts.

and can be used to measure work distribution accurately. As the ratios P_X/P_Y and S_X/S_Y show the strongest linear association, we considered both the ratios as a metric to determine equal contribution in terms of hours worked individually. In the remainder of the paper, we will refer to the ratios P_X/P_Y and S_X/S_Y as time-based ratios.

To answer RQ1, we considered each team's work ratio and time-based ratios for each assignment and the final project. Furthermore, we compared the work and time-based ratios to determine if both team members contributed equally in terms of lines of code and time spent. Moreover, we considered the total work ratio of each team to understand if the total work done by each team member in a team was equal throughout the semester. We did not consider grades, as team members received the same grade in each assignment, irrespective of their individual contributions.

In RQ2, we want to determine if providing students with a recommended collaborative workflow changes their work distribution. Hence, we recommended a git workflow on the second assignment but not on the first, as we wanted to measure how the work distribution changes due to the recommended workflow. We instructed students to list their contributions in a README file for the first assignment. Additionally, we informed them that we would not verify individual contributions when grading their work. The course staff graded their submission as a team, i.e., both members received the same score regardless of individual contribution. For the second assignment, as per the recommended workflow, we required teams to create and maintain different branches – an individual branch for each team member, a test branch for integrating changes from each individual branch, and a main branch for the final version. When grading, we assigned equal weights to individual branches and team branches. For example, if student X mentioned in the README that they implemented feature F1, we verified the correctness of F1 in branch X. If student Y mentioned that they implemented feature F2, we verified the correctness of F2 in branch Y. If F1 depended on F2 and vice-versa, we created dummy data to facilitate testing. Finally, we verified the correctness of features F1 and F2 in the main branch.

To answer RQ2, we compared the work ratio of teams in assignments 1 and 2. If the compared work ratios differ, the recommended workflow changes how students distribute work within a team. However, the recommended workflow had no effect if they were similar. Moreover, since we did not provide any guidance in the third assignment and the final project, we also compared their work ratios with the previous assignments to determine if the recommendation provided in the second assignment changed work distribution in later assignments where guidance was absent.

To answer RQ3, we used a similar strategy to understand teams' git usage. In the first assignment, where students did not receive a recommended git workflow, we determined how many teams used git branches to collaborate. We compared this with the teams' git workflows in the second assignment, where we gave students explicit instructions to create and maintain four branches – a main branch for production-ready code, a test branch to test integration, and two development branches; one for each team member. Moreover, we wished to determine if the workflow recommended in the second assignment persisted in future assignments. Therefore, we identified how teams used branches and merges in the third assignment and the final project. Our motivation for this analysis is to understand if students can learn and adopt advanced version control features

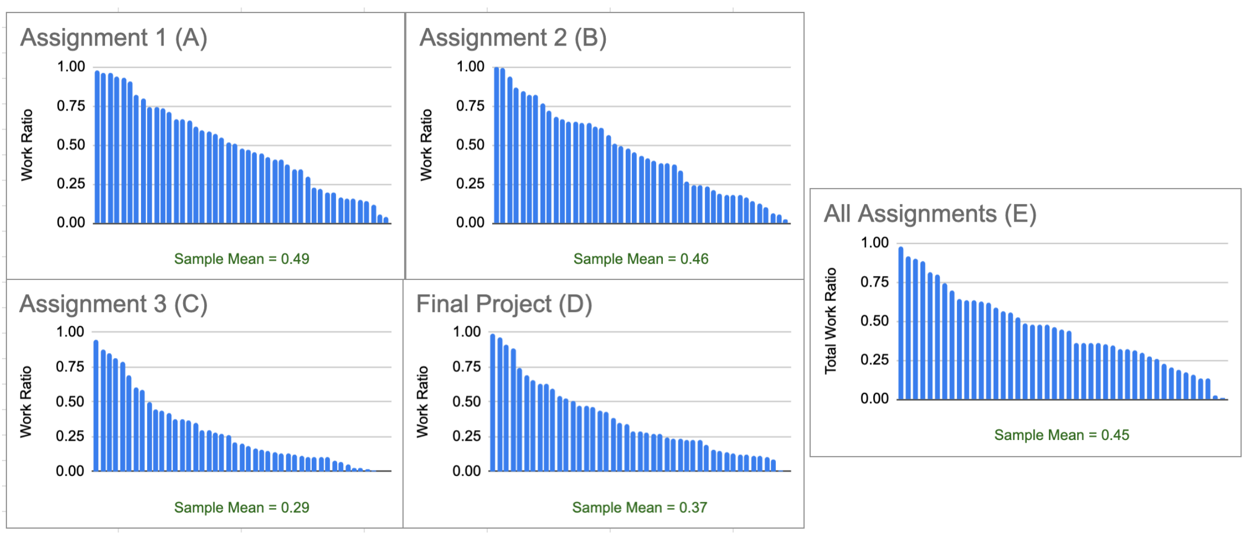

Figure 1: Figures A - D show the work ratio of all teams in assignments 1, 2, 3, and the final project. Figure E shows the total work ratio of all teams across all assignments and the final project. Each bar represents the work ratio of a team. The corresponding population mean estimate for each sample mean is not 1 at a significance level of 0.05.

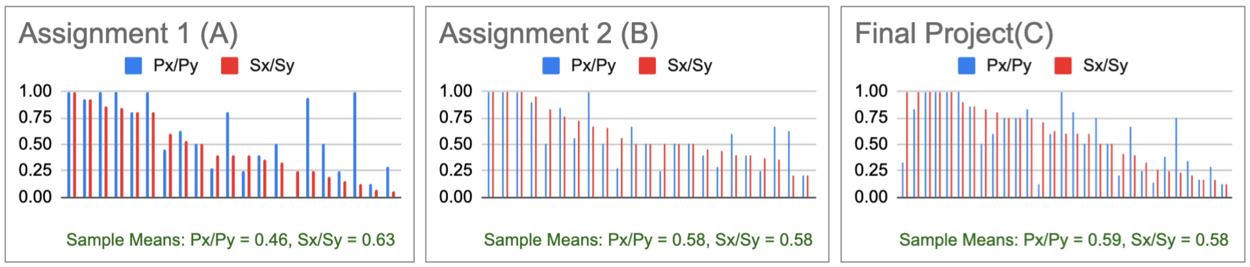

Figure 2: Time-based ratio based on the time reported by each team in assignments 1, 2, and the final project. Each bar represents the time-based ratio of a team. The corresponding population mean estimate for each sample mean is not 1 at a significance level of 0.05.

commonly used in collaborative software projects.

Results

RQ1 Results. In the first, second, and third assignments and the final project 8/45, 8/45, 5/45, and 4/45 teams had a work ratio of 0.75 or greater (see figures 1A-D). Moreover, 23/45, 26/45, 36/45, and 32/45 teams in assignments 1-3 and the final project have a work ratio of 0.5 or less. *Therefore, for most teams, one team member contributed two times or more lines of code than the other for each assignment.*

Furthermore, the third assignment and the final project have more teams with a work ratio of less

	P_X/P_Y vs. WorkRatio	S_X/S_Y vs. WorkRatio
Assignment 1	0.33*	0.28
Assignment 2	-0.12	0.2
Final Project	-0.008	0.26

Table 2: Comparison of work ratio and time-based ratios based on Pearson Correlation Coefficient for assignments 1,2, and final project. The coefficients with * indicate that they are statistically significant, assuming $\alpha = 0.05$ and a two-tailed t-test.

than 0.25 than the first two assignments. *The most likely reason for later assignments having a relatively more unequal distribution of work than earlier assignments is time constraints*. Towards the end of the semester, when teams work on the later assignments, workload from other courses may have led to a time crunch, which resulted in students with more coding experience shouldering more responsibility to finish the work on time.

The total work ratio of 6/45 teams is 0.75 or higher (Figure 1E). Moreover, 28/45 teams have a total work ratio of 0.5 or less. *Therefore, for 28/45 teams, one team member contributed two times or more lines of code than the other in the entire semester.*

Based on the reported time-based ratios from the survey, only 9/21 teams have $P_X/P_Y >= 0.75$, and 6/21 teams have $S_X/S_Y >= 0.75$ for the first assignment (see Figure 2A). Similarly, for the second assignment, 6 of 23 teams have $P_X/P_Y >= 0.75$ and $S_X/S_Y >= 0.75$ (see Figure 2B). For the final project only 9/29 teams have $P_X/P_Y = 0.75$ and $S_X/S_Y >= 0.75$ (see Figure 2C). Therefore, *members in the majority of the teams reported that they did not contribute equally in terms of hours worked individually*.

Comparing each team's work and time-based ratios using the Pearson Correlation Coefficient shows a weak linear association (see Table 2). Also, only the correlation between P_X/P_Y and the work ratio for the first assignment is statistically significant. The weak correlation between the objective metric (work ratio) and the subjective measure (self-reported time-based ratio) is consistent with prior research efforts. While this observation points to the limitation of the metrics, we note that both ratios for most teams across all assignments and the final project are low (< 0.75). *Therefore, despite the weak correlation, both time-based and work ratios indicate a similar trend, i.e., unequal work distribution between team members*.

RQ2 Results. Less than 50% (19/45) of the teams had a higher work ratio in the second assignment than in the first (see Figure 3). Moreover, only one team had a work ratio of 1, and 7 teams had a work ratio of 0.8 or more in the second assignment. Therefore, *based on the work ratio, recommending a collaboration workflow in the second assignment did not encourage students to contribute code equally*.

30/45 teams in the third assignment and 28/45 teams in the final project had a work ratio less than the second assignment (see Figure 3). Indeed, for most teams, the work ratio in the third assignment and the final project is lower than that of the second assignment. Therefore, *the recommended workflow in the second assignment did not cause students to contribute code more*

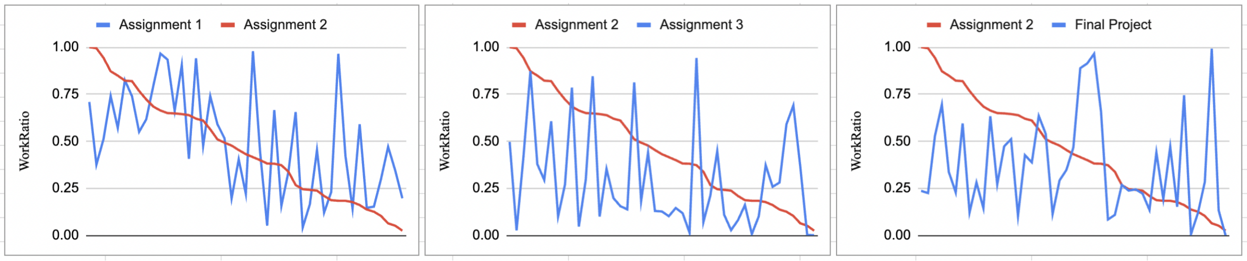

Figure 3: Comparison of work ratio of assignments 1, 3, and final project with that of assignment 2.

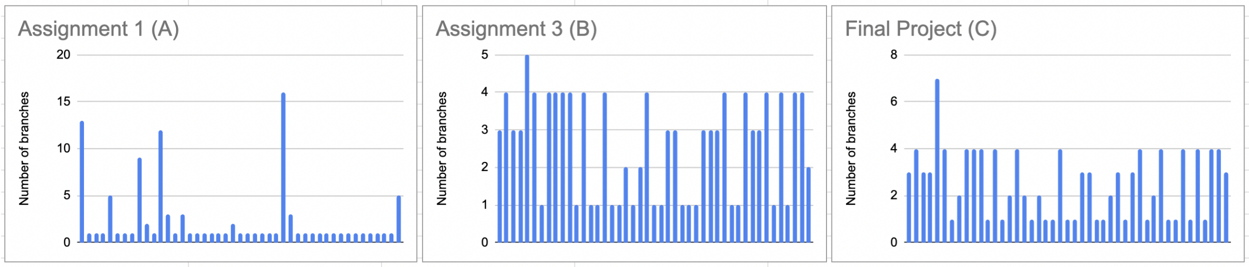

Figure 4: Number of branches used by each team in assignments 1, 3, and the final project. Each bar represents the time-based ratio of a team. Assignment 2 is not shown as all teams were required to use 4 branches.

equally in later assignments. Hence, recommending a git workflow did not make code contribution more equal than not recommending one.

RQ3 Results. Most of the teams (34/45) in the first assignment worked on the main branch (see Figure 4A), i.e., both team members committed and pushed code to the main branch without maintaining a separate branch. However, a few teams (3/45) followed a workflow where each team member created and maintained separate branches and used the main branch for integration. The remaining teams (5/45) created and maintained a branch to develop each individual feature and integrated all the features in the main branch. Therefore, *in the absence of explicit instructions/guidance, most students are unable to use the full power of a version control system to collaborate seamlessly.*

All teams created and maintained four branches in the second assignment, which is not surprising as we explicitly asked the students to maintain a main branch for production, a development branch for each team member where they develop and test the features they are responsible for, and a test branch to merge and test the integration of code from the development branches.

A third of the teams (15/45) used the workflow recommended in the second assignment in the third assignment, i.e., they created and maintained a main branch, a test branch, and two development branches (see Figure 4B). Further, 10/45 teams used a variant of the recommended workflow, using three branches - the main branch and two development branches. They used the main branch for both integration and production. The remaining teams used either only the main branch, the same as most teams in the first assignment, or two branches – the main branch and a development branch. Similarly, half the teams (22/45) used the recommended git workflow or a variant in the final project (see Figure 4C). The remaining teams used either only the main branch or a development branch and the main branch. Therefore, *a sizeable number of students found the workflow provided in the second assignment valuable as half or more of the teams continued using it even when they were not required to do so*.

Results

Observations. There are several potential reasons for the unequal work distribution of students working in a team even after explicit instructions were provided to encourage students to contribute equally.

1. Students in a team likely had different backgrounds in programming. Hence, code written by members with more programming experience may have made it to the repository at the expense of students with less experience. This contribution pattern is likely why few students reported equal hours as their partners even if they contributed fewer lines of code. Moreover, students with less experience may have committed less code to the team's repository due to a steeper learning curve. However, this is not necessarily a problem, as less experienced students can learn from the more experienced teammate even if they are not contributing the same amount of code. Moreover, this team dynamic mimics the real world, where a team comprises several individuals with different experience levels. On the other hand, less experienced students may get disillusioned if their more experienced teammate/s dismiss their efforts and opinions, especially when they are learning new concepts.
2. The students in a team wrote code together, most likely on the same device, similar to pair programming, with one member adding the bulk of the code to the repository. This style of collaboration is beneficial if the course staff switches the roles of coder and reviewer periodically to enable all team members to gain experience in writing, reviewing, and testing code. However, the absence of such role-switching may hamper some students from gaining experience in writing code.
3. In the second assignment, where we provided explicit guidelines, we graded the students based on the features they implemented on their branches (as mentioned by them in the README) and the main branch. We did not verify individual contributions via commit history. Therefore, it is possible that one team member wrote the bulk of the code on their branch, and the other member pulled/merged the changes onto their branch before merging the final version to the main branch. Consequently, both students in the team would have received the same grade despite having contributed unequally. Due to a lack of strong enforcement, students may have continued unequal contribution practices in subsequent assignments. A more stringent grading policy based on checking commit histories may incentivize students to distribute work equally.

The RQ3 results demonstrate that most students do not use advanced version control features such as branching and merging unless explicitly guided, as we did by providing a recommended workflow in the second assignment. The likely reason is that version control systems such as git have extensive documentation, which can overwhelm new users. Consequently, students get lost in the documentation and lose the motivation to learn the features needed for effective collaboration. However, when provided with instructions on recommended workflows and relevant commands, teams adopt them and continue to use them in the future.

Recommendations.

Based on the results and observations, we recommend the following action items for any course where students must work on programming assignments in a team using a version control system.

1. *Evaluate all students before forming teams*. Our results imply that allowing students to pick teams leads to unequal work distribution. Instructors should perform a formative assessment of students before dividing them into teams. The evaluation results will give the teaching staff a better idea of how to pair students. Prior efforts have proposed different strategies for grouping students [18]. Instructors should explore them and come up with an appropriate grouping method.
2. *Enforce equal work distribution*. Without enforcement, not all students are motivated to distribute work equally. Consequently, students who work more than others in a team are discouraged from collaborating, adversely impacting all students' learning outcomes. Instructors should measure students' individual and group contributions. Additionally, instructors can assign specific roles to students in a team, such as tester, designer, and implementer. Students should be evaluated according to contributions based on their assigned roles and the team's performance. For example, in a team of two students with assigned roles for testing and implementing, the individual evaluations will include measuring code coverage of tests, their effectiveness in detecting bugs in an incorrect implementation, and the correctness of the implementation based on student tests and instructor-defined tests. The roles should be switched in subsequent assignments.
3. *Provide detailed and explicit guidance on using version control*. While guidance on using version control systems for collaboration does not improve work distribution in teams, it allows interested students to learn and adopt an effective workflow early. Indeed, the RQ3 results demonstrate that most teams adopted and continued to use the recommended workflow. Therefore, instructors should provide all relevant information to use version control effectively before starting the programming projects/assignments. One way to ensure all students have the necessary information is to have them complete an assignment focused on using version control before proceeding.

Threats to Validity

We used lines of code added by each member to measure individual contributions. This metric might be limiting as a student may have contributed in other ways, such as resolving ambiguities in requirements, testing, documentation, design, or assisting teammates. However, we do not consider these since this paper measures individual contributions in writing code. Future studies should consider such aspects when measuring individual contributions from a broader perspective. For example, to measure the contribution of an assisting teammate, a mutual-aid factor, based on a specific format of git commit messages, can be considered [19].

We derived the time-based ratios from student responses in unsupervised surveys. As a result, students potentially colluded with each other to report inaccurate data. However, we analyzed the reported data using multiple metrics described in Section 3.2 to minimize the impact of misreporting on the results.

References

[1] L. Williams and R. L. Upchurch, "In support of student pair-programming," in *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 327–331. [Online]. Available: https://doi.org/10.1145/364447.364614

[2] C. McDowell, B. Hanks, and L. Werner, "Experimenting with pair programming in the classroom," *SIGCSE Bull.*, vol. 35, no. 3, p. 60–64, jun 2003. [Online]. Available: https://doi.org/10.1145/961290.961531

[3] S. Faja, "Evaluating effectiveness of pair programming as a teaching tool in programming courses," *Information Systems Education Journal*, vol. 12, pp. 36–44, 01 2014.

[4] A. Kirsh and I. Gaber, "Satisfaction, time investment and success in students' programming exercise," in *Proceedings of the Programming Experience 2016 (PX/16) Workshop*, ser. PX/16. New York, NY, USA: Association for Computing Machinery, 2016, p. 9–20. [Online]. Available: https://doi.org/10.1145/2984380.2984382

[5] J. Calver, J. Campbell, and M. Craig, "Student perspectives on optional groups," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 18–24. [Online]. Available: https://doi.org/10.1145/3545945.3569739

[6] E. Aivaloglou and A. v. d. Meulen, "An empirical study of students' perceptions on the setup and grading of group programming assignments," *ACM Trans. Comput. Educ.*, vol. 21, no. 3, mar 2021. [Online]. Available: https://doi.org/10.1145/3440994

[7] V. Farrell, G. Ravalli, G. Farrell, P. Kindler, and D. Hall, “Capstone project: Fair, just and accountable assessment,” in *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 168–173. [Online]. Available: https://doi.org/10.1145/2325296.2325339

[8] J. Porquet-Lupine and M. Brigham, “Evaluating group work in (too) large cs classes with (too) few resources: An experience report,” in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 4–10. [Online]. Available: https://doi.org/10.1145/3545945.3569788

[9] R. M. Parizi, P. Spoletini, and A. Singh, “Measuring team members’ contributions in software engineering projects using git-driven technology,” in *2018 IEEE Frontiers in Education Conference (FIE)*, 2018, pp. 1–5.

[10] J. J. Sandee and E. Aivaloglou, “Gitcanary: A tool for analyzing student contributions in group programming assignments,” in *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3428029.3428563

[11] N. Gitinabard, Z. Gao, S. Heckman, T. Barnes, and C. F. Lynch, “Analysis of Student Pair Teamwork Using GitHub Activities,” *Journal of Educational Data Mining*, vol. 15, no. 1, pp. 32 – 62, Mar. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7646845

[12] K. Buffardi, “Assessing individual contributions to software engineering projects with git logs and user stories,” in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 650–656. [Online]. Available: https://doi.org/10.1145/3328778.3366948

[13] J. Leinonen, L. Leppänen, P. Ihantola, and A. Hellas, “Comparison of time metrics in programming,” in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ser. ICER ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 200–208. [Online]. Available: https://doi.org/10.1145/3105726.3106181

[14] C. Hundhausen, P. Conrad, O. Adesope, and A. Tariq, “Combining github, chat, and peer evaluation data to assess individual contributions to team software development projects,” *ACM Trans. Comput. Educ.*, may 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3593592

[15] C. Hundhausen, P. Conrad, A. Carter, and O. Adesope, “Assessing individual contributions to software engineering projects: a replication study,” *Computer Science Education*, vol. 32, no. 3, pp. 335–354, 2022. [Online]. Available: https://doi.org/10.1080/08993408.2022.2071543

[16] StackOverflow, “Stack Overflow,” https://stackoverflow.com/, 2008, Accessed: 26-Jul-2023.

[17] GitHub, “GitHub Classroom,” https://classroom.github.com/, 2021, Accessed: 26-Jul-2023.

[18] R. Agrawal, B. Golshan, and E. Terzi, “Grouping students in educational settings,” ser. KDD ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1017–1026. [Online]. Available: https://doi.org/10.1145/2623330.2623748

[19] K. Tanabata, A. Hazeyama, Y. Yamada, and K. Furukawa, “Proposal of an evaluation method of individual contributions using the function point in the implementation phase in project-based learning of software development,” *Procedia Computer Science*, vol. 192, pp. 1524–1531, 10 2021.