2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)

2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) | 979-8-3503-2259-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-SEET58685.2023.00030

Improving Grading Outcomes in Software Engineering Projects Through Automated Contributions Summaries

Kai Presler-Marshall
Department of Computer Science
Bowdoin College
Brunswick, ME, US
Email: k.preslermarshall@bowdoin.edu

Sarah Heckman
Department of Computer Science
North Carolina State University
Raleigh, NC, US
Email: sarah_heckman@ncsu.edu

Kathryn T. Stolee
Department of Computer Science
North Carolina State University
Raleigh, NC, US
Email: ktstolee@ncsu.edu

***Abstract*—Teaming is a key aspect of most professional software engineering positions, and consequently, team-based learning (TBL) features heavily in many undergraduate computer science (CS) and software engineering programs. However, while TBL offers many pedagogical benefits, it is not without challenges. One such challenge is assessment, as the course teaching staff must be able to accurately identify individual students' contributions to both encourage and reward participation. In this paper, we study improvements to grading practises in the context of a CS1.5 introductory software engineering course, where assessing individual students' contributions to weekly lab assignments is done manually by teaching assistants (TAs). We explore the impact of presenting TAs with automated summaries of individual student contributions to their team's GitHub repository. To do so, we propose a novel algorithm, and implement a tool based off of it, AutoVCS. We measure the impact on grading metrics in terms of grading speed, grading consistency, and TA satisfaction. We evaluate our algorithm, as implemented in AutoVCS, in a controlled experimental study on Java-based lab assignments from a recent offering of NC State University's CS1.5 course. We find our automated summaries help TAs grade more consistently and provides students with more actionable feedback. Although TAs grade no faster using automated summaries, they nonetheless strongly prefer grading with the support of them than without. We conclude with recommendations for future work to explore improving consistency in contribution grading for student software engineering teams.**

***Index Terms*—grading consistency, program analysis, software engineering teams**

I. Introduction

Professional software engineering is, almost without exception, a team-based activity, drawing together multiple developers with disparate skills to solve complicated problems [1]. To help prepare students for this reality, most computer science (CS) and software engineering programs include some form of collaborative learning. Many programs explicitly use team-based learning (TBL) to give students a realistic collaborative experience and teach teaming skills [2], [3], [4].

TBL means that students must learn effective collaboration. One common challenge is encouraging students to participate equitably in their team's efforts, rather than freeriding off of the contributions of their teammates [5] or dominating the effort and thus preventing others from contributing [6]. Ensuring equitable contributions generally requires individual contribution grades for each student [7], which is often done by TAs.

Unfortunately, TAs may struggle to provide students with consistent and actionable feedback that recognises individual contributions. While providing a grade deduction can serve as a motivator to do better for freeriders, the most effective way for students to learn is by providing formative feedback with specific comments on how to improve [8], [9]. For students to improve, they must receive consistent feedback [10]. However, TAs may provide inconsistent grades and feedback [11], [12] that impedes learning.

Prior efforts to improve grading consistency have considered the use of rubrics [13], [14] and having multiple TAs or instructors grade each assignment [15], [14]. However, even with rubrics, grading individual contributions is subjective and difficult to do consistently [13]; replicating grading efforts can help resolve this, but is a drain on teaching staff resources. In this study, we consider if automation can take the place of multiple graders and offer similar benefits to grading consistency.

We frame our work around the following research questions:

- **RQ1**: *Can automated summaries of student contributions enable faster grading by TAs?*
- **RQ2**: *Can automated summaries of student contributions enable more consistent grading by TAs?*
- **RQ3**: *Can automated summaries of student contributions enable less frustrating grading from the perspective of TAs?*
- **RQ4**: *How do automated summaries of student contributions enable better feedback?*

To answer these questions, we designed an algorithm to summarise individual students' code contributions to team assignments, and built a reference implementation, AutoVCS, for Java projects tracked through GitHub. To attribute contributions to individual students, our algorithm uses Git commit history and abstract syntax tree (AST)-based differencing [16], and thus presents richer insights than tallying lines of code

979-8-3503-2259-0/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-SEET58685.2023.00030

(LOC) [17], [18]. We conduct a controlled experimental study with 13 current or former TAs to understand how they grade assignments when using automated summaries. Our results show that automated summaries can help TAs grade more consistently and provide more nuanced and actionable feedback, compared to grading without summaries from the algorithm and tool.

The contributions of this paper are as follows:

- an algorithm for summarising what individual developers have contributed to collaborative software assignments,
- an implementation of our algorithm as a tool, AutoVCS, which works on Java projects,
- a demonstration that our algorithm, implemented in AutoVCS, helps TAs grade lab assignments more consistently and provide students with more useful feedback, and
- a demonstration that TAs prefer grading with summaries from our algorithm.

II. Related Work

Here, we discuss teaming in computer science and software engineering education (Section II-A), formative assessment (Section II-B), data mining in software engineering (Section II-C), and measuring individual developers' team contributions (Section II-D).

A. Teaming in Computer Science and Software Engineering

Professional software engineering is a team-based activity [1], [19] and consequently teaming is a core outcome assessed by accreditation agencies [20]. To prepare students for this, teaming is a core component of many CS programs [21], [18], [22]. Teaming is used in many different contexts, ranging from algorithms courses [23] to introductory courses [24] to final capstones [21], [2]. Teaming is particularly common in software engineering programs or courses [3], [21], [18], [25], [2].

Prior work has demonstrated that not all teams of software engineering students work effectively together. Research suggests that students lack the time management and project management skills necessary to run a team [26], [4], resulting in general disorganisation or interpersonal conflicts [4], [27], [25]. Iacob and Faily [4] report that dysfunction is a risk in student software engineering teams, where low engagement or poor communication can hamper individual and team outcomes. They identify that there may be team dysfunction, but do not study its causes. Marques [27] proposes having a "monitor" conduct weekly meetings with teams of software engineering students, observing them work and providing feedback on the overall team function and contributions of each member. Presler-Marshall et al. [25] demonstrate that these struggling teams can be proactively identified with a collaboration reflection survey. Other prior work has indicated *social loafing*, or *freeriding*, as the dominant challenge in undergraduate engineering projects [5], [28]. Borrego et al. [28] argue that freeriding is particularly common in introductory courses and those with academically unbalanced teams [29], as these students struggle the most at identifying how each member can participate equitably. In this work, we aim to help instructors identify when students' contributions are insufficient and help them provide the students with more actionable feedback on what effective teamwork looks like.

B. Formative Assessment

Formative assessment is a key component of student-centred learning, where students are provided early, and regular, feedback on their work [9], [8]. Sadler argues that formative assessment helps students identify the characteristics of "high quality work" by providing them with a "direct authentic evaluative experience" [30]. This gives students the opportunity to better engage with material and identify what they have mastered and where they need to improve. While formative assessment may take many forms, "specific written comments are more effective than providing grades" at maximizing learning [31], [9]. Formative assessment has been demonstrated to be particularly helpful for lower-achieving students, helping them perform better and thus narrowing the gap between the lowest and highest achievers in a class [8].

Formative assessment is commonly used in CS and software engineering education [32], [33], and may take many different forms. Common approaches include giving students many, low-stakes assignments or exams [34], [35], [36] and pre-tests to assess areas for improvement [37]. Formative assessment may be used in collaborative learning, giving students a chance to engage in peer learning [35] or providing feedback that focuses on working in teams [38]. Peer evaluations can also serve as formative assessment [39].

C. Data Mining in Software Engineering

Prior work in data mining in software engineering focuses on version control systems (VCSs) such as GitHub. Most approaches consider large-scale analysis of open-source projects [40], [41], [42], [43], and often include machine learning approaches [44].

Prior work has explored data mining within computer science and software engineering education to understand how student teams work. Glassy [45] studies software engineering teams using SVN data and concludes that intermediate deadlines can counteract a tendency to procrastinate. Merle et al. [17] look at process and product features to see which features correlate with student grades on software engineering projects. They conclude that none work well, but LOC performs best. Gitinabard et al. [44] use Git data to create a machine learning classifier for identify types of commits from commit messages. Smith [46] presents gitRHIG for visualising information from assignments that use Git, although they merely demonstrate its features and provide little formal evaluation.

Evaluating the processes and practises followed by student teams can be a tedious experience, particularly when teams use multiple distinct tools (such as GitHub, Jenkins, and Heroku). To help instructors synthesise information across multiple sources, Garcia et al. [47] present BlueJay, a customisable tool that pulls together information from GitHub, Travis, Pivotal Tracker, and others into a series of dashboards. They report that these dashboards help instructors evaluate student development

processes, and the dashboards reveal that much work remains in encouraging students to follow better Agile development processes. By contrast, in our context Agile development processes are taught more extensively in a followup course, and thus our contributions summary algorithm, described in more detail in Section IV, focuses on the code contributions students make rather than the development processes that they follow.

To enable drawing richer conclusions, prior research has considered program analysis techniques. Fluri et al. [48] use AST analysis with ChangeDistiller, which builds ASTs representing two revisions of a Java file, and then performs tree differencing to identify what was changed between the two versions. Feist et al. [16] also use AST differencing to understand the evolution of open-source projects. Spirin et al. [49] present PSIMiner, a tool for producing annotated ASTs from source code, thus enabling richer analytics from it. Our algorithm builds on ChangeDistiller to identify changes made in Java code, and integrates this with Git commit history to understand how individual students are contributing to their team's overall effort.

D. Measuring Developer Contributions

Measuring individual developers' contributions to software projects remains an open challenge in industry and education. Most techniques build off version control systems such as Git and GitHub, measuring metrics including the number of commits [17], [18], [40], [50], [51], pull requests [18], [51], or lines of code [17], [18] made by each developer. However, all of these approaches are limited. Commits and pull requests can be large or small, and lines of code may represent substantive changes or simply reformatting existing code or committing auto-generated code. Measuring non-code contributions is particularly challenging and is typically done manually [19], [40]. In education, individual contributions are typically assessed manually, usually by TAs [7]. Software engineering projects in education also often use peer evaluations to give more feedback on who is contributing and how [52], [53].

III. Background

At NC State University, CS1.5 is a Java-based introductory software engineering course taken by all CS majors and minors and is open to non-majors. CS1.5 typically has between 250 and 350 students a semester, and is taught by one PhD professor and 12-18 TAs, giving a student:teaching staff ratio of approximately 20:1. CS1.5 teaches fundamentals in object-oriented design and development, best practises in software testing, and other core software engineering processes and practises; additionally it covers topics such as finite state machines and how to build and use linear data structures. To apply concepts covered in lecture, CS1.5 has a companion lab, where students work in teams of three to implement and test a Java application. Pair programming is introduced in the first lab session, and students are encouraged, but not required, to work collaboratively on lab tasks. Students work together on the same team, and use the same GitHub repository, for three or four weeks, at which point teams are scrambled for the next set of labs. Students complete a total of 11 labs with three different teams over the semester. When students rotate to a new team, they build off of the best implementation of their new teammates. In this way, students develop a medium-sized application, writing approximately 2,500 lines of code over 11 weeks. As described by Heckman and King [54], Jenkins is used to give students immediate feedback on their code and automate most grading.

With most grading automated through Jenkins, TAs are responsible only for grading Javadoc (to ensure it describes the code) and individual contributions (which are evaluated by checking commit history on GitHub to ensure each student is participating equitably). We have observed that grading individual contributions is a slow task that TAs dislike. Prior work has shown that even with rubrics, precisely evaluating contributions requires subjective judgement [13]; thus TAs are unlikely to draw meaningful single-point distinctions. Consequently, TAs provide coarse grades and feedback, giving a 0 ("No contributions"), 5 ("Insufficient contributions") or 10 ("Sufficient contributions"), typically with no further elaboration. However, this level of feedback may be insufficient for students to identify what a sufficient contribution looks like, and *why* their contribution was deemed insufficient. Prior work argues that students want more feedback on their work [55], and that providing this feedback can help improve the quality of their work [9]. In this study, we aim to identify whether our algorithm can assist TAs in providing students with this higher-quality feedback.

IV. Contributions Algorithm & AutoVCS

To identify whether automated contributions summaries can support grading, we developed an algorithm that uses commit histories and program analysis to summarise individual students' code contributions to team-based assignments. We then built a reference implementation, AutoVCS, which operates on assignments hosted on GitHub and written in Java. Our algorithm features three main steps; a full implementation is available in our GitHub repository [56].

1) **Metadata Extraction:** Metadata is extracted for each repository, storing commit hashes, dates and times for each commit, commit author, and a list of the files changed on each commit. This step is performed on Line 2 of Algorithm 1. While this information could be extracted from Git directly as part of the next step, AutoVCS extracts this information using the GitHub API and stores it locally in a database to improve the performance of subsequent steps and support user deduplication.[1]
2) **Change Extraction:** Similar to work done by Feist et al. [16], this step extracts changes made on each commit. It does so by traversing Git history to identify changed

[1]We have observed, similarly to Feist at al. [16], that many students will commit their work under multiple aliases, which otherwise impedes gaining a full picture of their contributions. Deduplicating aliases allows us to combine contributions made across aliases, and using a local database allows deduplicating these aliases much more efficiently than rewriting Git history.

Algorithm 1 Contribution Summary Algorithm

```
1: procedure SUMMARISECONTRIBUTIONS(repo,[...]) ▷ Computes a contributions summary for a Git repository, optionally within a time window, showing the contributions of each user
2:     RepoMetadata ←initRepository(repo)  ▷ Extract metadata, and optionally deduplicate users manually
3:     R1 ← clone(repo)
4:     R2 ← clone(repo)
5:     ContribsByCommit ← {}
6:     for commit C in RepoMetadata do
7:         if C.parent is null or C.isMergeCommit or C.isOutOfTimeWindow then
8:             continue
9:         end if
10:
11:        ContribsForCommit ← {}
12:        Check out R1 to C
13:        Check out R2 to C.parent
14:        for ChangedFile in C.ChangedFiles do
15:            AstNew ← buildAST(R1.ChangedFile)  ▷ Build an AST representing the new version of the file
16:            AstOld ← buildAST(R2.ChangedFile)  ▷ Build an AST representing the old version of the file
17:            ContribsForFile ← diff(AstNew, AstOld) ▷ Compute an edit script between ASTs to identify contributions
18:            ContribsForCommit.insert(ContribsForFile)
19:        end for
20:
21:        ContribsByCommit.insert(C, ContribsForCommit)  ▷ Map each commit to the changes made as part of it
22:    end for
23:    ByUser ← summarise(ContribsByCommit)  ▷ Summarise changes per-user, to show changes across files and commits
24:    return ContribsByUser
25: end procedure
```

files, and then building ASTs from adjacent file revisions and computing an edit script between them. This step is shown in Algorithm 1 from lines 6 to 22. AutoVCS uses our improved version of ChangeDistiller [48] to build and difference ASTs for each file.

3) **Contributions Summaries:** Detailed edit scripts for each file on each commit are aggregated to present higher-level summaries for each user; this is shown in Algorithm 1 on line 23, with more details shown in Algorithm 2. The resulting summaries are shown in Figure 1. Summaries are computed with three levels of granularity: Ⓘ a weighted [57] sum of all contributions; Ⓘ a summary of changes made across all files; and Ⓘ a summary of changes made to each file. Additionally, to allow for grading non-code contributions, a full list of commits can be shown (Ⓘ). For the Java code supported by AutoVCS, Ⓘ and Ⓘ use a condensed version of the change types proposed by Gall et al. [57]; Ⓘ is a weighted combination of these changes. Option Ⓘ also uses the weighted contribution scores to compute a percentage contribution for each member. Individual code changes are summarised into four categories: 1) changes to classes, 2) changes to methods, 3) changes to documentation, and 4) all other changes. These are shown in Ⓘ in Figure 1.

Algorithm 2 Summarise Changes By User Algorithm

```
1: procedure SUMMARISEBYUSER(ContribsByCommit) ▷ For a group of commits and associated fine-grained changes, presents a summary per user and a contribution score per user
2:     ContribsPerUser ← {}
3:     for (Commit, Contribs) in ContribsByCommit do  ▷ For each user, combine contributions
4:         if Commit.author not in ContribsPerUser then
5:             ContribsPerUser.insert(Commit.author, {})
6:         end if
7:         ContribsPerUser.insert(Commit.author, Contribs)  ▷ Add contributions from this commit to the running tally of contributions for this user
8:     end for
9:
10:    SummarisedContribs ← {}
11:    for User, Contribs in ContribsPerUser do  ▷ Summarise and weight contributions for each user
12:        UserContribScore ← 0
13:        UserContribSummary ← {}
14:        for Contrib in Contribs do
15:            UserContribSummary.insert(label( Contrib.type), existingCount+1)  ▷ Summarises detailed edit operations into higher-level contribution type
16:            UserContribScore += weight(Contrib.type) ▷ Computes weighted score for user based on type of contribution
17:        end for
18:
19:        SummarisedContribs.insert(User, {UserContribScore, UserContribSummary})
20:    end for
21:
22:    return SummarisedContribs
23: end procedure
```

Prior work has shown that LOC represents the best, although still not particularly good, predictor for team grades [17]. We hypothesise that part of the problem is that not all LOC changes are equal: languages such as Java contain substantial "boilerplate" code that is often auto-generated and thus does not represent a meaningful contribution. To address this, AutoVCS recognises four common boilerplate methods [58], [59] in Java code: `hashCode()`, `equals()`, getters, and setters.

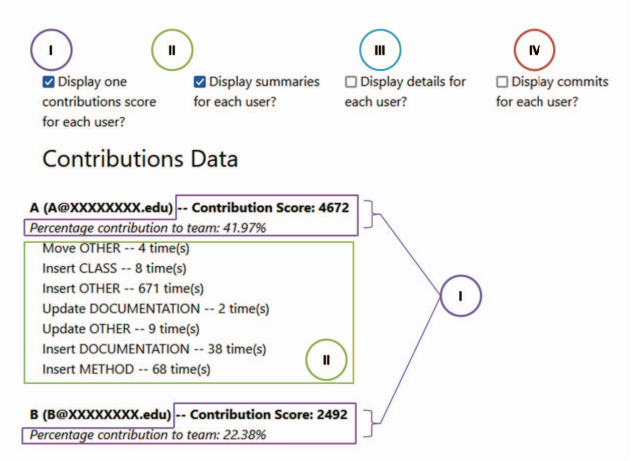


Fig. 1: A trimmed contributions summary produced by AutoVCS. All four types of summaries can be toggled on and off independently; two are enabled. For brevity, details for student B and all contributions for student C are not shown.

Changes to these methods are skipped so that contributions are not artificially increased by autogenerated code. In our course context, GUI files are provided by the teaching staff, so AutoVCS has a toggleable option to skip them. We hypothesise that these options, combined with that AST differencing implicitly ignores formatting changes, may offer better insights than just changed LOC or number of commits.

AutoVCS is a web application, and can be run in interactive mode and batch mode. In interactive mode, the user selects a single repository and time window and summaries are computed live and displayed. Batch mode instead runs from a JSON configuration file, and produces an HTML summary page for each repository specified. In this mode, the summary pages are self contained and do not depend upon AutoVCS' application server, and thus can be used externally (for example, hosted on GitHub Pages). On a Xeon E5-2670 with 16GB RAM and an SSD, analysis takes about two minutes for a repository with 100 commits. Batch mode also supports creating summaries for multiple repositories in parallel to improve performance; details are in our GitHub repository [56].

V. Study

To answer our research questions, we designed and conducted a two-part controlled experimental study with 13 participants to evaluate whether our algorithm, as implemented in AutoVCS, can help TAs grade more effectively (RQ1 & RQ2), make grading less frustrating (RQ3), or provide better feedback (RQ4). The study outline and research questions answered in each part of the study are shown in Figure 2. We recruited participants from two groups: a) 44 students who have served as TAs for two team-based undergraduate software engineering courses (CS1.5 and a third-year advanced software engineering course) within the past two years, and b) all CS PhD students at NC State University with TA experience. Nine students from group a) and four students from group b)

Fig. 2: Study Outline, showing the parts of the study, the approximate time spent on each part, and what RQs were answered by each.

participated. The participants had an average of 6.6 years of experience with Java (median: 6) and 4 semesters of TA experience (median: 4). Four participants identified as female, and five as members of a minority racial group.

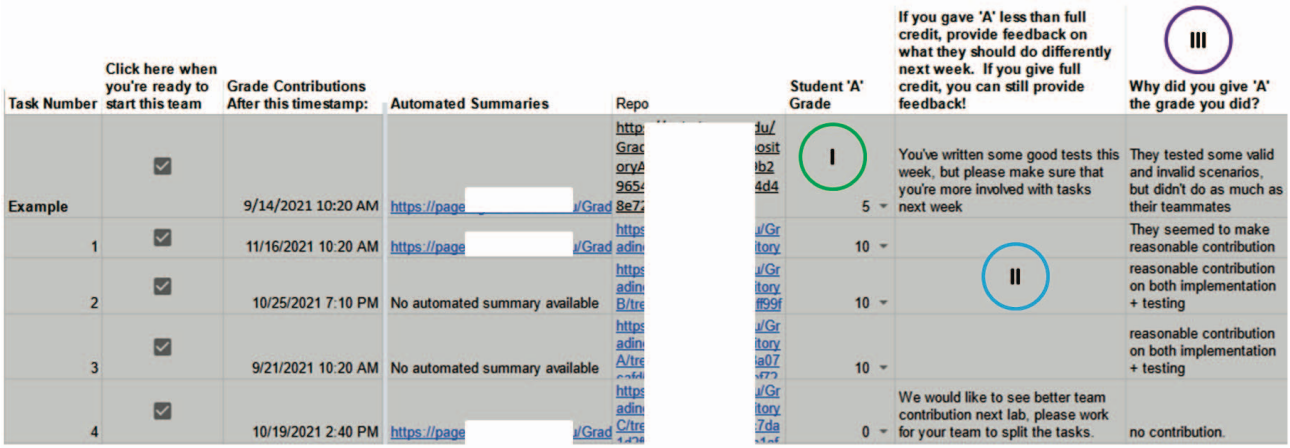
This study was conducted in four, two-hour lab sessions, held physically in a computer lab. We provided snacks, but participants were not otherwise compensated. All sessions followed the same procedures, and participants attended only one session. An outline of the study is shown in Figure 2. A brief introduction was provided to all participants before Part 1, explaining both parts of the study, the format of the tasks, and how to use the automated contributions summaries. In Part 1 (Section V-B), participants graded lab assignments from a recent offering of our CS1.5 course. Some assignments were graded without automated assistance (the *control* group) and some with summaries from our algorithm and AutoVCS (the *experimental* group). In Part 2 (Section V-C), participants evaluated feedback from other participants in the study. Finally, at the end of the study, participants completed a brief reflection (Section V-D).

A. Terminology

We refer to the individual students whose assignments were graded as *subjects*. We refer to the 13 TAs who participated in our study as *participants* or *raters*, depending on context. We refer to the grades assigned by *raters* to *subjects* as *ratings*. We refer to *rating subjects* when referring to individual students, or *grading assignments* when referring to the entire three-person team.

B. Part 1: Grading

In Part 1, participants were tasked with grading 17 lab assignments from a recent offering of our CS1.5 course. We provided each participant a Google Sheets spreadsheet, where each assignment to grade was on a separate row (Figure 3). A random subset of nine assignments had summaries from AutoVCS (the *experimental* group); the other eight had no summaries available (the *control* group). For each assignment, the spreadsheet contained a) a link to the GitHub repository with the code, b) a reminder of the time interval to grade, and c) where applicable, a link to the contributions summary from AutoVCS. The order of the 17 tasks was randomised for each participant. For each subject on each team, raters provided a) a contribution score (0, 5, or 10, as discussed in Section III), b) if the score was not a 10, a comment to the subject explaining what to do differently to receive more points, and c) a comment, not shared with the subject, giving the rationale for the score. We instrumented the spreadsheet to reveal tasks one at a time and capture start times and end times for each task. An excerpt



Task Number	Click here when you're ready to start this team	Grade Contributions After this timestamp:	Automated Summaries	Repo	Student 'A' Grade	If you gave 'A' less than full credit, provide feedback on what they should do differently next week. If you give full credit, you can still provide feedback!	Why did you give 'A' the grade you did?
Example	☑	9/14/2021 10:20 AM	https://page…/Grad	http…/Grac…oryA…965…8e7…	5	You've written some good tests this week, but please make sure that you're more involved with tasks next week	They tested some valid and invalid scenarios, but didn't do as much as their teammates
1	☑	11/16/2021 10:20 AM	https://page…/Grad	https…adin…	10		They seemed to make reasonable contribution
2	☑	10/25/2021 7:10 PM	No automated summary available	https…adin…B/tre…	10		reasonable contribution on both implementation + testing
3	☑	9/21/2021 10:20 AM	No automated summary available	https…adin…A/tre…	10		reasonable contribution on both implementation + testing
4	☑	10/19/2021 2:40 PM	https://page…/Grad	https…adin…C/tre…	0	We would like to see better team contribution next lab, please work for your team to split the tasks.	no contribution.

Fig. 3: Excerpt from the spreadsheet used for Part 1, showing grades (①), comments (②), and rationale (③). Feedback for students B and C is not shown.

from the spreadsheet used, showing grades and comments for several students, is shown in Figure 3.

The assignments to grade were prepared by anonymising 18 weekly lab assignments from a recent offering of our CS1.5 course, replacing authors in Git commits and code with pseudonyms.[2] These anonymised assignments were hosted on GitHub Enterprise to mirror normal grading practises. One assignment was used as an example to demonstrate the tasks and the contributions summaries; the other 17 were used as tasks for the study.

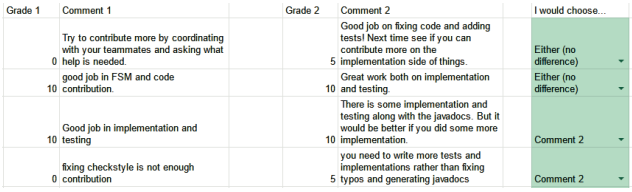
C. Part 2: Evaluating Feedback

In Part 2, participants were asked to put themselves in the mindset of a student receiving feedback and evaluate its actionability. We provided participants with ten pairs of grades and comments from other participants in the study and asked them to choose which comment from each pair is "*more helpful in letting you know what to improve upon*", or *Either (no difference)* as appropriate. One grade and comment in each pair came from an assignment from the control group, and one came from the experimental group. This label was not shown, and the order of the two comments was randomised. An example of the spreadsheet used is shown in Figure 4.

D. Reflection Survey

Finally, we asked participants to complete a brief reflection on the grading experience. The reflection asked participants how they used the automated summaries; how helpful they found each of the main features (shown in Figure 1); how they would improve the summaries; and whether they would choose to use them again. We also collected basic demographic information.

[2]Rewriting files in Git while keeping the history (commit author and timestamps) is not officially supported, and could not be performed on repositories where students introduced merge conflicts. Eighteen assignments could be completely anonymised.



Grade 1	Comment 1	Grade 2	Comment 2	I would choose...
0	Try to contribute more by coordinating with your teammates and asking what help is needed.	5	Good job on fixing code and adding tests! Next time see if you can contribute more on the implementation side of things.	Either (no difference)
10	good job in FSM and code contribution.	10	Great work both on implementation and testing.	Either (no difference)
10	Good job in implementation and testing	10	There is some implementation and testing along with the javadocs. But it would be better if you did some more implementation.	Comment 2
0	fixing checkstyle is not enough contribution	5	you need to write more tests and implementations rather than fixing typos and generating javadocs	Comment 2

Fig. 4: Excerpt from the spreadsheet used for Part 2, showing several pairs of comments alongside corresponding votes.

E. Data Description

1) Survey Responses: We converted the text Likert scale to numbers, and treat them as interval-scaled data [60], where 1 maps to the lowest score (e.g., "Not at all helpful" and "Never"), and 5 maps to the highest score (e.g., "Extremely helpful" and "Always"). We qualitatively analysed the open-ended questions. All 13 participants completed the reflection survey.

2) Data Details: In Part 1, we tasked 13 participants with grading 17 assignments. Some participants graded more slowly than others, and consequently not all participants finished grading all assignments. The 13 participants in our study graded a total of 204 assignments, providing 610 ratings for individual students.[3]

In Part 2, we provided each participant with ten pairs of grades and comments and asked them to choose which comment from each pair was more actionable. The 13 participants chose comments from the *experimental* group 60 times, comments from the *control* group 43 times, and expressed no preference 27 times.

3) Analysis: To answer RQ1 we compared grading times for assignments from the *control* and *experimental* groups. The

[3]Note this is not an exact multiple of 3, as two participants each missed rating one subject. If all participants had graded all assignments, there would have been 663 ratings (13 participants * 17 assignments * 3 students per assignment = 663 ratings).

TABLE I: Grading time, in minutes, for assignments that were graded manually (*Control*) or with automated summaries (*Experimental*). Times are split into the first eight assignments (*first half*) and second nine assignments (*second half*) graded by each participant.

	First Half		Second Half		Overall	
	Mean	Median	Mean	Median	Mean	Median
Control	7.57	7.13	4.36	4.02	5.85	5.31
Experimental	7.88	6.79	4.66	4.10	6.35	5.44

distribution of the elapsed times is skewed right; consequently, we chose a Mann-Whitney U test, a non-parametric test.

To answer RQ2, we computed inter-rated reliability (IRR) for the *control* and *experimental* groups. Raters provided three grades for each assignment (one for each of the three students on the team); thus each student is a unique subject rated by up to 13 raters. We use Krippendorff's Alpha [61] for computing IRR, as it handles a different number of ratings per subject. As suggested by Zapf et al. [62] we calculated confidence intervals for both groups and identified the *q-value* where they are disjoint.

To answer RQ3, we read through responses to the reflection to understand how participants used the summaries and whether they would choose to use them again.

We answered RQ4 with two different metrics. To identify if automated summaries help participants provide better feedback we performed Fisher's Exact Test on the preferences from Part 2. To understand if automated summaries help TAs see nuance and discern between partial and full contributions, we performed a test of two proportions on the rates of partial credit for each of the two groups.

VI. Results

In this section we present results for whether automated contributions summaries can enable faster (Section VI-A) or more consistent (Section VI-B) grading. We also consider impacts on the grading experience (Section VI-C) and feedback to students (Section VI-D).

A. RQ1: Grading Speed

We find no significant difference in grading speed from automated summaries. As shown in Table I, participants graded assignments in the *control* group in an average of 5.85 minutes. Participants graded assignments in the *experimental* group in an average of 6.35 minutes. A Mann-Whitney U Test confirmed that the difference was not significant ($p = .677$).

We observe a learning curve as participants get more comfortable with, and consequently faster at, grading assignments. To evaluate a learning curve, we compared times taken to grade the *first half* and *second half* of the assignments within the *control* and *experimental* groups using Mann-Whitney U tests. We observed learning effects in both groups, showing that regardless of how participants graded assignments, they got faster over time ($p < .001$ for both groups).

We observe that grading may *feel* faster when using automated summaries. One participant reflected "*it definitely felt faster to grade*" with the automated summaries. While the numbers do not back this up, if the process feels faster, TAs may consider it less of a burden.

RQ1: *Automated summaries do not impact grading speed.*

B. RQ2: Grading Consistency

We find that automated summaries can help TAs grade more consistently. We calculated Krippendorff's Alpha (α) separately for the *control* and *experimental* groups to identify how consistently the raters of each subject agreed with each other. We find $\alpha = 0.286$ for the *control* group, and $\alpha = 0.609$ for the *experimental* group. At $q = .021$[4], the confidence intervals are disjoint, showing that automated summaries significantly improve grading consistency.

We note, however, that even though automated summaries help TAs grade more consistently, $\alpha = 0.609$ still indicates a relatively low level of agreement. Krippendorff argues that "*it is customary to require* $\alpha \geq .800$" [65], which participants in our study did not meet. We discuss possible causes of this and implications in Section VII-B.

RQ2: *TAs grade assignments more consistently using automated summaries than without them.*

C. RQ3: Grading Preferences

We find that TAs have a strong preference for grading with automated summaries. All 13 participants said that they would prefer to use automated summaries for grading in the future, with 11 participants saying they would *strongly prefer* them. Participants gave the automated summaries an average rating of 4.85 out of 5. One participant said "*I think the tool was a huge help*" and rated it 5 out of 5. Another participant, who rated the summaries 4 out of 5, said they were "*very straightforward to use*".

As shown in Figure 5, participants find all of the features of the contributions summaries to be useful. As discussed in Section V-E1, we calculated the average score given to each feature. Participants found the *List of commits for each user* (Ⓥ in Figure 1) to be the most useful feature, rating it 4.46/5 (halfway between *Very Helpful* and *Extremely Helpful*). *Percentage Contribution to Team* (part of ① in Figure 1) was rated as the second most useful, with a score of 3.92/5 (corresponding to *Very Helpful*). All features received a rating of 5 from at least one participant, and received an average rating of at least 3.4/5, approximately halfway between *Moderately Helpful* and *Very Helpful*. Our results thus show that both summaries of commit history and more advanced program analysis can assist with grading.

RQ3: *TAs strongly prefer grading with summaries from AutoVCS and find all of the features helpful.*

[4]Krippendorff's α uses *q-values* as opposed to *p-values* as they provide improved resilience when performing multiple comparisons. *q-values* are interpreted the same way as *p-values* [63], [64].

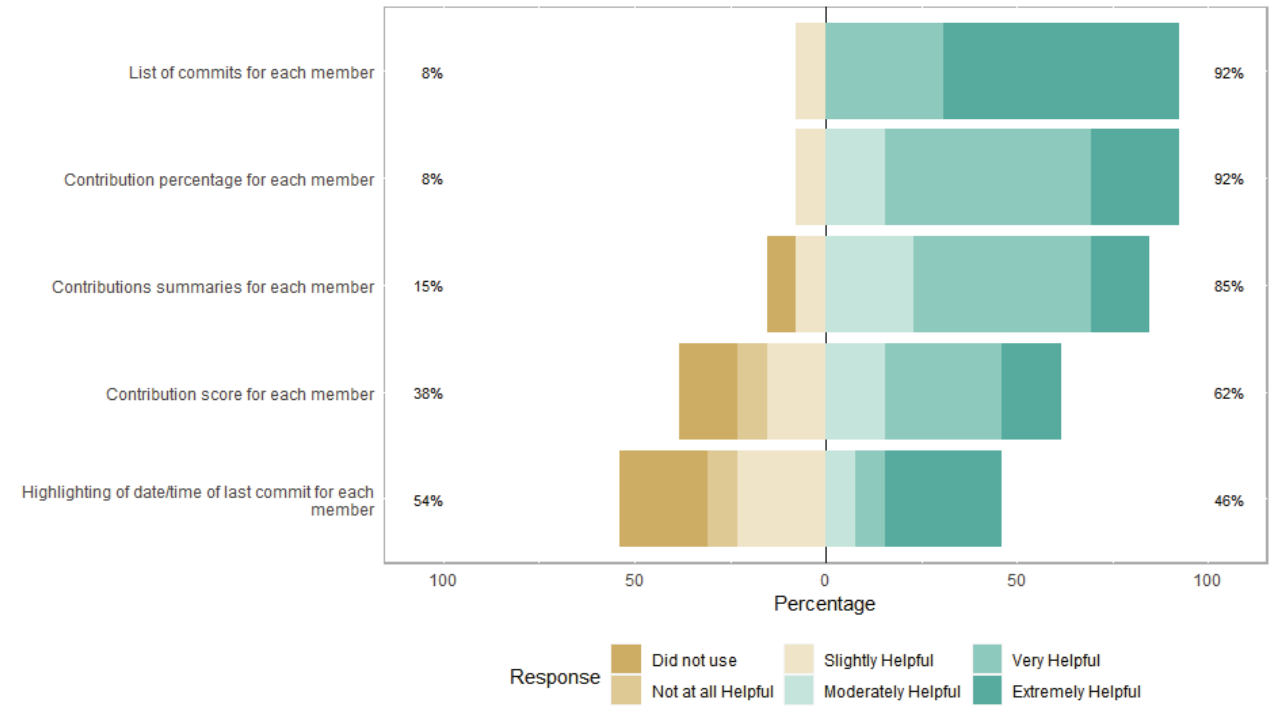


Fig. 5: Ratings from the participants in our study for each of the main features of our contributions summaries algorithm and AutoVCS.

D. RQ4: Feedback Quality

TAs consider feedback from assignments graded with automated summaries to be more actionable than assignments graded manually. As described in Section V-C, we asked participants to choose between pairs of comments from other participants in the study to choose which one makes it clearer "*what to improve upon*". A Fisher Exact Test confirmed ($p = .0311$) a preference for comments from the *experimental* group, thus showing that TAs consider feedback from their peers more useful when it came from assignments graded with automated summaries.

We also observe that the quantity of feedback is impacted by automated summaries. We compared the rate at which partial credit was assigned in both groups, and find TAs award partial credit to 17.1% of subjects in the *control* group, and 24.9% of subjects in the *experimental* group. A test of two proportions shows that this difference is significant ($p = .018$). TAs are expected to provide feedback on where to improve alongside partial credit, but are not required to do so for full credit. By giving more partial credit, this may help TAs provide students with more feedback and thus improve learning outcomes [31], [9].

One participant reflected that "*I was terrified at how much [the automated summaries] made me reconsider some of my initial grading thoughts*". We thus see evidence that our contributions summary algorithm may be able to help TAs grade more carefully, and thus provide better feedback and improved learning outcomes [31], [9]. We expand further on this in Section VII-A.

RQ4: *TAs consider feedback from assignments graded with automated summaries to be more helpful than feedback from assignments graded without them, and automated summaries help TAs see nuance and provide partial credit more often.*

VII. Discussion

In this section, we consider how TAs used the automated summaries when grading (Section VII-A), probe grading inconsistencies (Section VII-B), discuss threats to validity (Section VII-C) and explore future work (Section VII-D).

A. TA Use of Automated Summaries

Prior work has shown that, paradoxically, improving the quality of an AI or machine learning algorithm may result in a net negative overall performance impact, as the humans responsible for reviewing the output instead choose to defer to the machine over offering their own judgement [66], [67]. We were curious to see if the high-level contributions scores, and corresponding percentage contributions (① in Figure 1), would have a similar impact on our participants: TAs who might see a high (or low) overall contributions score, and look no further before deciding what grade to give. Our results suggest that this did not happen. We observed that TAs spent the same amount of time grading assignments when they had the automated contributions summaries to support them as they did grading

assignments without the automated summaries. Although we expected, and observed, a learning curve as TAs learn how to use the automated summaries and more broadly gain familiarity with the study tasks, we observed no statistically-significant difference in the learning effect between groups: grading times in the *control* and *experimental* groups improved at the same rate. This matches against our results in Section VI-D, namely by showing that the automated summaries helped TAs see nuance between different students' submissions and award *more* partial credit rather than less. This confirms that the improved consistency seen in Section VI-B was not a result of giving more subjects no credit, or full credit. Thus, our results suggest that there was *not* an over-reliance on the automated summaries, and the TAs used them to guide their grading rather than serving as a rubber stamp upon the numbers produced by our algorithm and our tool.

B. Improving Grading Consistency

In Section VI-B, we found that automated summaries improve grading consistency, but that consistency remains an issue. To probe this further, we focused on the most extreme cases: subjects who were given full credit (10) and no credit (0) by different raters. We found seven of these subjects within each of the *control* (*manual grading*) and *experimental* (*automated summaries*) groups. To understand these ratings, we read through the comments and rationale from each rater for these subjects.

We find that while the number of these disagreements does not differ across both groups, the causes do. In the *control* group, four of the seven disagreements came from issues *identifying* individual contributions. In two cases, a rater gave credit even when the subject had made no contributions. In an additional case, a rater gave credit for work done outside of the time window (work for a different lab, which used the same repository) and in a final case, a rater missed contributions that were made by the subject. By contrast, in the *experimental* group, we saw only two issues with *identifying* contributions. In one case, the rater gave credit for contributions outside the time window; in the second, the rater appeared to miss contributions within the time window[5]. The remaining cases (3 from the *control* group, 5 from the *experimental* group) were caused by disagreements over what contributions deserved credit (non-code contributions, such as system testing and documentation) and cases of pair programming. The sample size is small, but these results suggest that automated summaries may help TAs more accurately *identify* individual students' contributions; work remains to ensure that students are credited equitably for these contributions.

C. Threats to Validity

Conclusion: To combat any impacts of multiple study sessions, we used a script to introduce the study procedures and ensure that the experience was comparable for all participants.

[5]As discussed in more detail in Section VII-D, participants appeared to struggle the most with identifying non-code contributions, such as project management tasks and system testing.

Differences in elapsed times between groups were calculated with nonparametric tests to handle skewed data. Grading consistency, or inter-rater reliability, was calculated using Krippendorff's Alpha, which handles missing data [61].

Internal: To counter learning effects, the order of tasks for each participant was randomised.

Participants knew that their behaviour was being studied, and thus may have graded more carefully than they would otherwise do. However, this applies to participants in both the *control* and *experimental* groups equally, and a significant improvement was still observed.

Construct: We measure consistency by calculating inter-rater reliability, evaluating whether TAs' ratings agree with each other. We do not consider whether the ratings agree with an expert, such as a course instructor. However, our evaluation matches typical grading practises.

External: We conducted Part 1 of study using Google Sheets, and participants graded student labs from a recent semester. Both the study tasks and format emulate the normal grading experience. However, all assignments came from a single semester of a single course. We suggest future work to consider broader course contexts.

All participants in the study were current or former TAs for team-based computer science courses and have experience with evaluating individual contributions. However, some but not all of the TAs have worked on CS1.5 and thus are more familiar with the specific tasks in this study. Additionally, all participants volunteered to take part in the study, and thus may be more interested in evaluating tools that could improve their workflow, and more inclined to do a careful job, than the average TA. Busy graduate students who are less enthusiastic about their TA duties may grade assignments differently, possibly including relying more heavily on the automated summaries, than we observed.

As students, the participants are also familiar with interpreting feedback; however, all are graduate students or upper-level undergraduates, and consequently may do so differently from CS1.5 students.

D. Future Work

As discussed in Section VI-B, our results show that automated summaries can help TAs rate subjects significantly more consistently, but consistency is still relatively poor. Rubrics have been widely used to improve grading consistency and fairness [68], [69]; however, to the best of our knowledge, no prior work has evaluated their impact on assessing individual contributions. We propose evaluating whether rubrics can be used to help grade individual contributions more consistently.

We find that participants particularly struggled with grading pair programming. Pair programming is encouraged, as prior work has demonstrated its pedagogical benefits [70]; however, contributions in Git appear only under the name of the student who committed the code. While we instruct students to document pair programming via commit messages, it is not clear how often they do so. Unfortunately, students regularly forget to add `@author` tags to their Javadoc, which

means we lack a ground truth for a post-hoc analysis of which contributions resulted from pair programming, and thus for evaluating how effectively pair work was documented and assessed. Future work remains in studying how best to encourage students to document collaborative work, and then ensure that pair programming is graded fairly.

Much work remains to be done in account for non-code contributions. We found several participants who missed students' project management or system testing contributions, which are done in text or PDF documents instead of Java code. Prior work suggests automatically crediting non-code contributions is an open problem [19], [40]. We propose future work to support grading with automation in this area, particularly in semi-structured formats such as GitHub Issues or Pull Requests.

In Section VI-D we found that TAs consider feedback from assignments that were graded with automated summaries to be more actionable than feedback from manually-graded assignments. We suggest future work to evaluate learning gains by putting the feedback directly in front of students in the target course to evaluate whether these benefits transfer to an undergraduate student population.

In Section VII-A, we observed that TAs use the information provided in the automated summaries to get a more nuanced look at the student assignments, giving better feedback and more partial credit (Section VI-B and Section VI-D) than the control group. We propose future work to study the TAs' feedback in more detail, to determine how closely linked the feedback is to the automated summaries themselves. This work could inform further improvements to the automated summaries, to better help the TAs in finding the information they need for grading assignments, or potentially open up the possibility for auto-generating some or all of the feedback that is provided to students on their work.

In this paper, we performed a lab study to evaluate whether contributions summaries from our algorithm, as implemented in AutoVCS, can help TAs grade assignments more consistently and provide students with better feedback. While our results show a statistically significant improvement in grading consistency, we suggest future work to evaluate whether these benefits transfer to a larger set of assignments in a full course.

As described in Section V-B, all grading was performed in Google Sheets spreadsheets. This was done to mimic normal grading processes for the course. We propose future work to study how to support different grading workflows, including integrating contributions summaries into LMS platforms such as Moodle and Canvas.

VIII. Conclusion

In this work, we developed an algorithm for summarising individual students' code contributions to team assignments using Git commit history and AST analysis. We built a tool, AutoVCS, that implements our algorithm, and evaluated it with a controlled, A/B experimental study with 13 TAs, who graded some assignments with automated summaries and some assignments without them. We found that automated summaries help TAs grade assignments more consistently and provide students with feedback that is possibly more actionable. Additionally, although the contributions summaries do not help TAs grade assignments more rapidly, TAs nonetheless strongly prefer to grade assignments using them and would choose to use them again. Finally, we reflect on ways to further improve grading consistency with the use of rubrics, and suggest future work to explore the use language-agnostic contributions analysis and automated support for evaluating non-code contributions.

Acknowledgments

This work was supported in part by NSF SHF grants #1749936 and #1525173. We would like to thank the students of NC State University's Software Development Fundamentals course for allowing us to use their data for analysis, and the teaching assistants who volunteered their time and participated in our study. Finally, we would like to thank Dr. Dan Harris for providing his statistical expertise.

Data Availability

AutoVCS is open-source and is available with documentation and setup instructions on GitHub [56].

References

[1] I. Richardson, V. Casey, F. McCaffery, J. Burton, and S. Beecham, "A process framework for global software engineering teams," *Information and Software Technology*, vol. 54, no. 11, pp. 1175 – 1191, 2012.

[2] R. Bates, J. Hardwick, G. Salivia, and L. Chase, "A project-based curriculum for computer science situated to serve underrepresented populations," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE 2022. ACM, 2022, p. 585–591.

[3] J. E. Sims-Knight, R. L. Upchurch, T. A. Powers, S. Haden, and R. Topciu, "Teams in software engineering education," in *32nd Frontiers in Edu.*, vol. 3, 2002.

[4] C. Iacob and S. Faily, "The impact of undergraduate mentorship on student satisfaction and engagement, teamwork performance, and team dysfunction in a software engineering group project," in *Proceedings of the 51st ACM SIGCSE*, ser. SIGCSE '20. ACM, 2020, p. 128–134.

[5] D. Hall and S. Buzwell, "The problem of free-riding in group projects: Looking beyond social loafing as reason for non-contribution," *Active Learning in Higher Education*, vol. 14, no. 1, pp. 37–49, 2013.

[6] K. Presler-Marshall, S. Heckman, and K. T. Stolee, "What makes team[s] work? a study of team characteristics in software engineering projects," in *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, ser. ICER '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 177–188.

[7] A. Tafliovich, A. Petersen, and J. Campbell, "On the evaluation of student team software development projects," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. ACM, 2015, p. 494–499.

[8] P. Black and D. Wiliam, "Inside the black box raising standards through classroom assessment," vol. 80, 09 2010.

[9] B. Wisniewski, K. Zierer, and J. Hattie, "The power of feedback revisited: A meta-analysis of educational feedback research," *Frontiers in Psychology*, vol. 10, 2020.

[10] D. L. Butler and P. H. Winne, "Feedback and self-regulated learning: A theoretical synthesis," *Review of Educational Research*, vol. 65, no. 3, pp. 245–281, 1995.

[11] N. Glazer, "Formative plus summative assessment in large undergraduate courses: Why both?" *The International Journal of Teaching & Learning in Higher Education*, vol. 2014, pp. 276–286, 03 2015.

[12] J. Hayes, T. Lethbridge, and D. Port, "Evaluating individual contribution toward group software engineering projects," in *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003, pp. 622–627.

[13] A. Jonsson and G. Svingby, "The use of scoring rubrics: Reliability, validity and educational consequences," *Educational Research Review*, vol. 2, no. 2, pp. 130–144, 2007.

[14] I. Albluwi, "A closer look at the differences between graders in introductory computer science exams," *IEEE Transactions on Education*, vol. 61, no. 3, pp. 253–260, 2018.

[15] K. Raman and T. Joachims, "Methods for ordinal peer grading," in *Proceedings of the 20th ACM SIGKDD*, ser. KDD '14, 2014, p. 1037–1046.

[16] M. D. Feist, E. A. Santos, I. Watts, and A. Hindle, "Visualizing project evolution through abstract syntax tree analysis," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, 2016, pp. 11–20.

[17] K. Mierle, K. Laven, S. Roweis, and G. Wilson, "Mining student cvs repositories for performance indicators," in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ser. MSR '05. ACM, 2005, p. 1–5.

[18] 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 (FiE)*, 2018, pp. 1–5.

[19] F. Ramin, C. Matthies, and R. Teusner, "More than code: Contributions in scrum software engineering teams," in *Proceedings of the IEEE/ACM 42nd International. Conference on Software Engineering Workshops*, ser. ICSEW'20. ACM, 2020, p. 137–140.

[20] "Criteria for accrediting engineering programs, 2021 – 2022," Oct 2020. [Online]. Available: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2021-2022/

[21] J. Khakurel and J. Porras, "The effect of real-world capstone project in an acquisition of soft skills among software engineering students," in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, 2020, pp. 1–9.

[22] C. Hundhausen, A. Carter, P. Conrad, A. Tariq, and O. Adesope, "Evaluating commit, issue and product quality in team software development projects," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM, 2021, p. 108–114.

[23] X. Lin, J. Connors, C. Lim, and J. Hott, *How Do Students Collaborate? Analyzing Group Choice in a Collaborative Learning Environment*. ACM, 2021, p. 212–218.

[24] J. E. Burge, G. Gannod, M. Carter, A. Howard, B. Schultz, M. Vouk, D. Wright, and P. Anderson, "Developing cs/se students' communication abilities through a program-wide framework," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. ACM, 2014, p. 579–584.

[25] K. Presler-Marshall, S. Heckman, and K. Stolee, "Identifying struggling teams in software engineering courses through weekly surveys," *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, 2022.

[26] B. Oakley, R. Brent, R. Felder, and I. Elhajj, "Turning student groups into effective teams," *Journal of Student Centered Learning*, vol. 2, 01 2004.

[27] M. R. Marques, "Monitoring: An intervention to improve team results in software engineering education," in *Proceedings of the 47th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '16. ACM, 2016, p. 724.

[28] M. Borrego, J. Karlin, L. D. McNair, and K. Beddoes, "Team effectiveness theory from industrial and organizational psychology applied to engineering student project teams: A research review," *Journal of Engineering Education*, vol. 102, no. 4, pp. 472–512, 2013.

[29] V. Pieterse and L. Thompson, "Academic alignment to reduce the presence of 'social loafers' and 'diligent isolates' in student teams," *Teaching in Higher Education*, vol. 15, no. 4, pp. 355–367, 2010.

[30] D. R. Sadler, "Formative assessment and the design of instructional systems," *Instructional science*, vol. 18, no. 2, pp. 119–144, 1989.

[31] J. Hattie and H. Timperley, "The power of feedback," *Review of educational research*, vol. 77, no. 1, pp. 81–112, 2007.

[32] M. Marchisio, T. Margaria, and M. Sacchet, "Automatic formative assessment in computer science: Guidance to model-driven design," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2020, pp. 201–206.

[33] L. Benotti, M. C. Martnez, and F. Schapachnik, "A tool for introducing computer science with automatic formative assessment," *IEEE Transactions on Learning Technology*, vol. 11, no. 2, pp. 179–192, 2018.

[34] P. Belleville, S. A. Wolfman, S. Bradley, and C. Heeren, *Inverted Two-Stage Exams for Prospective Learning: Using an Initial Group Stage to Incentivize Anticipation of Transfer*. ACM, 2020, p. 720–738.

[35] S. MacNeil, C. Latulipe, and A. Yadav, "Learning in distributed low-stakes teams," in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ser. ICER '15. ACM, 2015, p. 227–236.

[36] H. Erdogmus, S. Gadgil, and C. Péraire, "Introducing low-stakes just-in-time assessments to a flipped software engineering course," 01 2019.

[37] M. C. Parker and Y. S. Kao, "How do you know if they don't know? the design of pre-tests in computing education research," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE 2022. ACM, 2022, p. 1147.

[38] S. Reckinger and B. Hughes, *Strategies for Implementing In-Class, Active, Programming Assessments: A Multi-Level Model*. ACM, 2020, p. 454–460.

[39] H.-J. Lee and C. Lim, "Peer evaluation in blended team project-based learning: What do students find important?" *Educational Techology and Society*, 2012.

[40] J.-G. Young, A. Casari, K. McLaughlin, M. Z. Trujillo, L. Hébert-Dufresne, and J. P. Bagrow, "Which contributions count? analysis of attribution in open source," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 242–253.

[41] I. Kwan, A. Schroter, and D. Damian, "Does socio-technical congruence have an effect on software build success? a study of coordination in a software project," *IEEE ToSE*, vol. 37, no. 3, pp. 307–324, 2011.

[42] A. Mauczka, F. Brosch, C. Schanes, and T. Grechenig, "Dataset of developer-labeled commit messages," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 490–493.

[43] D. M. German, "An empirical study of fine-grained software modifications," *Empirical Software Engineering*, vol. 11, no. 3, p. 369–393, sep 2006.

[44] N. Gitinabard, R. Okoilu, Y. Xu, S. Heckman, T. Barnes, and C. F. Lynch, "Student teamwork on programming projects. what can github logs show us?" in *Proceedings of the 13th International Conference on Educational Data Mining, EDM 2020*. International Educational Data Mining Society, 2020.

[45] L. Glassy, "Using version control to observe student software development processes," *Journal of Computing Sciences in Colleges*, vol. 21, no. 3, p. 99–106, feb 2006.

[46] C. M. Smith, "A toolset for mining github repositories in educational software projects," Ph.D. dissertation, 2018.

[47] C. García, A. Guerrero, J. Zeitsoff, S. Korlakunta, P. Fernandez, A. Fox, and A. Ruiz-Cortés, "Bluejay: A cross-tooling audit framework for agile software teams," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2021, pp. 283–288.

[48] B. Fluri, M. Würsch, M. Pinzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 725–743, 2007.

[49] E. Spirin, E. Bogomolov, V. Kovalenko, and T. Bryksin, "Psiminer: A tool for mining rich abstract syntax trees from code," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 13–17.

[50] N. Sviridov, M. Evtikhiev, and V. Kovalenko, "Tnm: A tool for mining of socio-technical data from git repositories," in *2021 18th ACM MSR*, 2021.

[51] M. V. Bertoncello, G. Pinto, I. S. Wiese, and I. Steinmacher, "Pull requests or commits? which method should we use to study contributors' behavior?" in *2020 IEEE 27th SANER*, 2020, pp. 592–601.

[52] B. Pérez and A. L. Rubio, "A project-based learning approach for enhancing learning skills and motivation in software engineering," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '20, 2020, p. 309–315.

[53] S. Brutus and M. B. L. Donia, "Improving the effectiveness of students in groups with a centralized peer evaluation system," *Academy of Management Learning & Education*, vol. 9, no. 4, pp. 652–662, 2010.

[54] S. Heckman and J. King, "Developing software engineering skills using real tools for automated grading," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: ACM, 2018, p. 794–799.

[55] J. Li and R. D. Luca, "Review of assessment feedback," *Studies in Higher Education*, vol. 39, no. 2, pp. 378–393, 2014.

[56] K. Presler-Marshall, "Autovcs." [Online]. Available: https://github.com/AutoVCS/AutoVCS
[57] H. C. Gall, B. Fluri, and M. Pinzger, "Change analysis with evolizer and changedistiller," *IEEE Software*, vol. 26, no. 1, pp. 26–33, 2009.
[58] A. Kamalizade, "How to reduce java boilerplate code with lombok."
[59] Changyi, "Java programming skills – boilerplate code." [Online]. Available: https://www.alibabacloud.com/blog/java-programming-skills-boilerplate-code_598058
[60] S. E. Harpe, "How to analyze likert and other rating scale data," *Currents in Pharmacy Teaching and Learning*, vol. 7, no. 6, pp. 836–850, 2015.
[61] K. Krippendorff, "Computing krippendorff's alpha-reliability," 2011.
[62] A. Zapf, S. Castell, L. Morawietz, and A. Karch, "Measuring inter-rater reliability for nominal data – which coefficients and confidence intervals are appropriate?" *BMC Medical Research Methodology*, vol. 16, no. 1, p. 93, Aug 2016.
[63] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.
[64] J. D. Storey, "The positive false discovery rate: a Bayesian interpretation and the q-value," *The Annals of Statistics*, vol. 31, no. 6, pp. 2013 – 2035, 2003.
[65] K. Krippendorff, *Content analysis: An introduction to its methodology*. SAGE, 2004.
[66] Z. Buçinca, M. B. Malaya, and K. Z. Gajos, "To trust or to think: Cognitive forcing functions can reduce overreliance on ai in ai-assisted decision-making," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, apr 2021.
[67] V. Lai and C. Tan, "On human predictions with explanations and predictions of machine learning models," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*. ACM, jan 2019.
[68] J. Feldman, *Grading for Equity: What It Is, Why It Matters, and How It Can Transform Schools and Classrooms*. SAGE Publications, 2018.
[69] K. Ragupathi and A. Lee, "Beyond fairness and consistency in grading: The role of rubrics in higher education," in *Diversity and inclusion in global higher education*. Palgrave Macmillan, Singapore, 2020, pp. 73–95.
[70] L. A. Williams and R. R. Kessler, "Experiments with industry's "pair-programming" model in the computer science classroom," *Computer Science Education*, vol. 11, no. 1, pp. 7–20, 2001.