Analysis of Student Pair Teamwork Using GitHub Activities

Niki Gitinabard
Allobee Inc.
niki.gitinabard@gmail.com

Sarah Heckman
NC State University
sarah_heckman@ncsu.edu

Collin F. Lynch
NC State University
cflynch@ncsu.edu

Zhikai Gao
NC State University
zgao9@ncsu.edu

Tiffany Barnes
NC State University
tmbarnes@ncsu.edu

Few studies have analyzed students' teamwork (pairwork) habits in programming projects due to the challenges and high cost of analyzing complex, long-term collaborative processes. In this work, we analyze student teamwork data collected from the GitHub platform with the goal of identifying specific pair teamwork styles. This analysis builds on an initial corpus of commit message data that was manually labeled by subject matter experts. We then extend this annotation through the use of self-supervised, semi-supervised learning to develop a large-scale annotated dataset that covers multiple course offerings from a second-semester CS2 course. Further, we develop a series of predictive models to automatically identify student teamwork styles. Finally, we compare trends in students' performance and team selection for each teamwork style to see if any of them reflected better student outcomes or different trends of help-seeking among students. Our analysis showed that applying self-supervised semi-supervised methods helps us to label larger subsets of data automatically and maintains and even sometimes improves the performance of the fully supervised models on a held-out validation set. Our analysis also showed that members of teams in which all members have significant contributions tend to have better performance in class, but their help-seeking behaviors are not significantly different.

Keywords: teamwork, GitHub, undergraduate, self-supervised Learning, help-seeking

1. Introduction

As teamwork has become a crucial part of many undergraduate CS courses, analyzing the patterns in students' teamwork becomes increasingly important both to monitor students' work and to scaffold good collaborative practices. Teamwork is common in professional settings and offering students the chance to engage in teamwork through their learning gives them the chance to gain important experience that can transfer to a professional setting (Feichtner and Davis, 1984; Reid and Wilson, 2005). However, students, particularly beginner students, are unfamiliar with the concept of teamwork and can struggle with collaboration (Feichtner and Davis, 1984). While instructor supervision and occasional interventions may help to smooth the students' teamwork experience, it can be difficult for instructors to offer such support due to large class sizes and the often irregular work schedules that students keep. In order to provide effective guidance it is necessary for instructors to understand how students work in teams, what behaviors support good learning, and how those behaviors are exhibited in immediate and observable features such as code commits rather than delayed formats such as self-evaluations. Automated models of collaborative practices can be used to support instructors in triaging student teams, guide immediate feedback, and support insights into other essential behaviors including help-seeking.

Consistent with *Social Learning Theory*, we believe that in collaborative learning contexts all team members should make significant contributions to the project in order to derive a benefit from the interaction. Social Learning Theory, as defined by Bandura and Walters (1977), highlights the four significant principles for successful collaborative learning: attention, reproduction, retention, and motivation. In their view, some of the principal benefits of collaborative learning come through examples provided by ones' peers. Similar to worked examples of problem solving skills (e.g. Van Gog and Kester (2012)), these peer examples can provide efficient opportunities for students to learn problem solving skills. According to Bandura and Walters however, the benefit of these examples is mediated by the extent to which students pay *attention* to the work of their peers and their ability to *reproduce* it; as well as the extent to which they *retain* those examples for later use and are *motivated* to commit to the work. Teams that are fractious, unbalanced, or where one student monopolizes the work can be demotivating, and can deny all members the opportunity to learn from their peers. Moreover, even teams that partition the work equally but do not share actions, may be less effective because they deny even committed peers the chance to learn from one another. Consequently, in order for students to benefit from peer work they should maintain an equal exchange of effort that avoids isolation. Thus, guided by Social Learning Theory we hypothesize that students participating in team projects where each member makes significant contributions to the project will perform better, not only in that specific project, but ultimately in the class as a whole.

In this research, we seek to address this hypothesis by analyzing students' contributions to and performance in CS2 courses that rely on multiple team projects. Prior research in Software Engineering has identified different ways to measure project contributions. Parizi et al., for example, defined measures such as the total number of commits, total additions, and the amount of time spent and provided those to the course instructors for performance assessment of individual students in team projects (Parizi et al., 2018). While these metrics are easy to calculate, the authors argue that all of them should be taken into account, as well as the difficulty of the subtask that the team member is working on, which requires manual evaluation of these metrics by an expert. However, as Lima et al. (2015) noted, project leaders believe that evaluating participation in projects is time-consuming and has no established criteria. As a result, we first need to define different teamwork styles and automatically categorize student teams to evaluate how students performed in different team environments. In our analysis, we chose to focus on students' commit behaviors and help-seeking. While prior research on student teaming (e.g. Perera et al. (2009)) has shown that ticketing behavior can be a useful indicator of team structure in advanced classes with longer-term projects, the students in our courses make no use of the ticketing features in their development environment, nor do they consistently use other platforms for organization. The students in these courses are new to CS and the use of ticketing is not covered in this or prior classes. As such the shared commits represent the best record of their coordination.

Using the commit messages we defined three teamwork styles based in part on the work of Coman et al. (Coman et al., 2014). The first type of teams works on different parts of the projects together, with all of the team members involved in each of the different activities such as implementation and testing. Coman et al. described this type of *collaborative teaming* as cases where the team members focus on the shared goals and work together to achieve them. The second type of teams partitioned the work by type of task, with all members making significant contributions and thus, each member specializing in one activity. One commonly observed case among such teams was those where one member engaged in feature development and another authored test cases. Coman et al. called this *cooperative teaming* where members have distinct sub-goals and support one another as needed (Coman et al., 2014). In the cases that we observed, students defined such goals based on the different parts of the grading rubric such as implementing features or covering the code by test cases. While this kind of work division may be productive, it can also indicate unhealthy team dynamics such as a lack of trust between team members over the quality of the work or unbalanced interactions that exclude some members from implementation decisions (Salomon and Globerson, 1989; van der Duim et al., 2007). This behavior may also indicate that students have, or believe they have, special skills or experience in some areas and not in others. While this may be positive as students bring their talents, to bear it may also mean that some team members fail to gain experience with crucial skills such as test writing or design. In many structured collaboration arrangements such as Process Oriented Guided Inquiry Learning (POGIL) or Pair Programming, this kind of division is incorporated. However, students are directed to rotate in such settings to allow all members gain equal experience. Finally, in our class, as in many other project-based courses, we observed a third more concerning pattern of *solo-submitters* where one member submitted the majority of the work on GitHub and the other member did little or none (less than 10%). This pattern is also sometimes termed as having a ”free-rider” (Salomon and Globerson, 1989; van der Duim et al., 2007). Prior research has shown that a relative balance of contributions in a team can lead to better team performance and higher member satisfaction (Seers, 1989). While it is possible that the team members did communicate offline and share work equally via other means, these teams did not practice effective collaboration via the version control system, which is a learning goal for the course. For this study, we did not differentiate between solo-submitters teams and those that communicated offline, because we did not have information about their offline activities.

For our analysis, we used data from six offerings of a CS2 course on Java programming for CS majors. More information on this course and the covered topics are mentioned in the Dataset section. We analyzed students' GitHub behavior to first, classify their commits into the different aspects of the project work such as "implementing features" or "testing" using the information in the commit messages. We then defined several features based on their number of commits and amount of code changes in each aspect of the work to classify students' teamwork styles into one of the three teamwork styles of "Collaborative", "Cooperative", or "Solo-Submit". Some examples of the defined features are "percentage of implementation commits" by each team member or the "number of testing code lines" submitted by each member. Finally, we analyzed and compared the teams in each category based on their member performance and help-seeking to understand the differences between these teamwork styles.

We can organize our work into the following four steps:

- S1. Automatically classify student commit messages into the different kinds of activities involved in projects using commit message texts and self-supervised learning.
- S2. Use the features defined based on commit texts and student activities to identify Collaborative, Cooperative, and Solo-submitting teams by self-supervised learning.
- S3. Analyze and compare the performance of the Students in Collaborative, Cooperative, and Solo-submit teams.
- S4. Analyze and compare the help-seeking behaviors of the students in Collaborative, Cooperative, and Solo-submit teams.

The findings of this study will help us to better understand student teamwork in early programming courses and how that teamwork interacts both with the team formation, student's help-seeking, and changes in the course structure. This work will also advance our understanding of how to model teaming behavior with minimal annotation. This will help facilitate future work on the design of effective team interactions in courses and the development of student- and instructor-facing scaffolds for team projects.

2. Background

Prior researchers have analyzed student habits in programming projects and identified patterns that can distinguish between higher and lower-performing students (Ahmadzadeh et al., 2005; Uchida et al., 2002; Spacco et al., 2015; Lin et al., 2016; Vihavainen et al., 2013; Carter et al., 2015; Carter et al., 2017; Blikstein, 2011; Watson et al., 2014; Hosseini et al., 2014; Chao, 2016). Research has shown that lower-performing students spend more time on solving programming problems (Ahmadzadeh et al., 2005; Uchida et al., 2002; Spacco et al., 2015; Murphy et al., 2009), while higher performers typically analyze the code more logically, and spot issues much faster (Uchida et al., 2002; Lin et al., 2016). Some visualization tools are also designed and used to analyze these activity patterns and make suggestions to the students such as Retina tool (Murphy et al., 2009). More recent work has focused on logs from version control systems used by students to analyze their behavior (Glassy, 2006; Reid and Wilson, 2005; Murphy et al., 2009; Mierle et al., 2005). While these studies provide a great insight into students' study and work habits, they are mainly focused on their individual work and consider neither the role they play in a team nor the impact of team behaviors on students' learning.

At the same time there has long been interest in analyzing students' teamwork behaviors, particularly in CS courses, to understand this impact (Näykki et al., 2014; Van den Bossche et al., 2006; Barr et al., 2005; Oakley et al., 2004; Wen et al., 2017; Feichtner and Davis, 1984; Lee, 2009). However, much of this work has used student surveys and instructor evaluations and very few look into their online activities (Kim et al., 2012; Liu et al., 2004; Ganapathy et al., 2011; Seers, 1989). Notable exceptions are the work of Seers who examined the equitability of student contributions and argued for balanced teams, and the work of Kay et al. (Kay et al., 2006; Perera et al., 2009) who conducted a detailed analysis of group work in an advanced CS course using analysis of students' engagement with version control, ticketing systems, and shared documentation. Informed by the Big 5 theory Kay et al. sought to identify patterns of behavior that were associated with desirable traits such as leadership and mutual performance monitoring (Perera et al., 2009) and to develop visualizations that could inform facilitators (Kay et al., 2006). Their research focused on advanced students who participated in teams of 5-6 over a 12-week course. As a consequence, their students were better prepared for group work and faced different challenges with interaction. The work we describe here, by contrast, is focused on students in early stage courses who interact over a shorter period of time and primarily make use of a single observable platform, version control.

While group work is an important component of learning at all levels students in introductory courses, like the ones we examine here, can be ill-prepared for it. Several issues can arise from this inexperience (Feichtner and Davis, 1984). These include some members "ganging up" in the team and excluding others from the work or members being "free-riders" (Salomon and Globerson, 1989; van der Duim et al., 2007) or "social loafers" who do not participate in the work. One pattern often observed in student teams with diverse skill sets is that higher performers may believe that they work better than their teammates and as a result, they may give up on collaboration (Lee et al., 2017). Indeed some students often referred to as "lone wolves" prefer to work alone from the start and their inclusion in a team can have a negative impact on all the members and the project outcome (Barr et al., 2005). Many of these negative patterns may be visible if we assess member contributions regularly. Past researchers have sought to evaluate this balance using a range of methods including cross-platform analysis of student engagement (Perera et al., 2009), counting total contributions (Hoegl and Gemuenden, 2001; Seers, 1989), or using richer methods including self-assessment surveys and video recording (Perera et al., 2009; Näykki et al., 2014). These latter approaches, while informative, are often time-consuming and impractical in larger courses yielding delayed information at best.

Some prior research on student teaming has also taken place in courses where the instructors grade the students' teamwork (Main and Sanchez-Pena, 2015; Northrup and Northrup, 2006; Imbrie et al., 2005) following performance models in industry (Parizi et al., 2018; Nguyen and Chua, 2016; Lima et al., 2015). While these evaluations include different aspects of work and are often reliable, they can also be subjective, difficult to automate, and time-consuming (Lima et al., 2015). Further, they are ill-suited for interventions as they are completed well after the interactions take place. This is also true of peer evaluations which are used in larger courses but which typically occur at the end of the projects (Van den Bossche et al., 2006; Wen et al., 2017; Feichtner and Davis, 1984; Lee, 2009; Lee et al., 2017; Oakley et al., 2004; Perera et al., 2009). These peer reviews can often be unreliable, particularly for assessing students' relative contributions (Feichtner and Davis, 1984). As a consequence, these evaluations are ill-suited either for monitoring ongoing projects in larger classes or for supporting any early intervention to improve team behaviors before it is too late.

Kim et al. and Liu et al. defined several metrics to track student activities and progress while working on team projects (Kim et al., 2012; Liu et al., 2004). These metrics were extracted from version control system logs and built on basic metrics including key events such as document creation, number of editors and edits, and final length (Kim et al., 2012), the total number of revisions, and average work days (Liu et al., 2004). They suggested these reports to assist instructors in regular team evaluations and not as a stand-alone tool. More recent studies have moved towards automated approaches such as extracting students' share of work from Version Control Systems (Ganapathy et al., 2011; El Asri et al., 2017). Ganapathy et al. (2011) used the number of documents edited by several group members as a measure to assess team collaborations and found that teams with more involved members tend to get a better outcome. While El Asri et al. (2017) used the number of commits and lines of code for measuring contributions and showed that these measures significantly predict member contributions. Thus, work on the automated evaluation of teamwork quality is relatively new and such evaluations are still far from being completely automated. In this work, we focus on taking the automated analysis of teamwork a step further and more hands-off.

3. Dataset

The work reported here is based upon data from six offerings of a CS2 Java Programming course at NC State for majors during fall semesters from 2015 to 2020. Topics covered in this course include object-oriented design, testing, composition, inheritance, state machines, linear data structures, and recursion. This course was offered in person by two instructors from 2015 - 2019 with some small sections being offered online. In the Fall of 2020, this course was moved online due to the COVID pandemic. Prior to 2020 all of the course offerings included two midterm exams (Tests 1 and 2), one final exam, multiple lab sessions, and 2-3 larger programming projects. In Fall 2020, the three exams were replaced by 15 quizzes that were given online over the course of the semester. The students in this year were permitted to re-take the quizzes as needed to improve their grades. For the sake of consistency however, we use their first quiz grades in our analysis before re-takes.

During 2015 - 2018, the course included three programming projects, the first of which was done individually, the second could be done individually or in a team of two, and the final was completed as a pair. While the project details varied somewhat over the years the difficulty of each project remained comparable. All of the Project 2 assignments in our dataset required students to implement some form of a Finite State Machine (FSM) containing at least five major classes, two I/O classes, and six state classes. Project 3, on the other hand, was centered on implementing and applying linked lists to manage data. The project required them to implement 6-7 major classes, two I/O classes, and five data structure-related classes. Project 2 tasks typically required 1200-1300 lines of code to complete while Project 3 required over 1400. In 2019 the number of projects in the course was reduced to two, with one being completed individually and one as a team. In all offerings, the students were permitted to request specific teammates or to have one assigned by the instructors. When teams were assigned the instructors used students' prior individual work (exams and individual projects) to pair individuals with similar skill levels. For all of the team projects, the students were first required to draft a high-level design of the system and a test plan individually before they are assigned to teams. If a student fails to submit their individual work they will not be assigned to a team and must complete the full task alone. The final implementation and testing are done based on a design provided by the instructor. More information about these courses is included in Table 1.

Students in this course use the Eclipse IDE for coding tasks and their projects were graded based on the teaching staff and their own test cases, code coverage from their test cases (Jacoco), maintaining specific coding style (SpotBugs, PMD, CheckStyle), and documentations (JavaDoc) (Heckman and King, 2018). The students also used Piazza as a discussion forum as well as GitHub version control system to share a code-base with their teammates and to submit their code for grading. When the students made changes to the code, they were able to store the changes by "committing" their code. In GitHub, making a commit means generating a checkpoint with a label explaining the changes called a "commit message". Then, the students uploaded these changes to the GitHub server by "pushing" the commits.

Table 1: Per-Class statistics in our dataset.

Class	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**
Total Students	181	206	203	247	248	278
Teaching Assistants	9	9	9	11	14	17
In-person Instructors	2	2	2	2	2	1
Remote Instructors	2	2	2	2	2	2
Average Grade	79.7	79.9	79.48	79.96	81.1	71.9
Project 2 pairs	39	46	58	49	69	71
Total Project 2 pair programming commits	0	89	10	10	2	109
Project 3 pairs	76	90	88	109	-	-
Total Project 3 pair programming commits	0	81	8	13	-	-
Avg commits per repository	110	43	59	56	68	56
Max commits per repository	325	209	339	247	377	264

Student repositories on GitHub were connected to a Jenkins server, which evaluated their code every time they pushed new changes. Jenkins is a continuous integration system that in this case is set up to download the current version of the code every time it is pushed and run the teaching staff test cases on the code. The system uses a web-based platform to show the students immediate feedback about the number of failed tests and the topic of them. Students cannot see the exact test cases. However, some of the hints include the test scenario so that students can write their own version of the teaching staff test. This was done more frequently in more recent years. The students could submit their codes to get feedback as often as they needed. In these classes, the team repositories had between 11 and 317 commits with an average of 122.8 and a median of 120.5. In this study, we focus on the students' activity logs on GitHub to automatically classify their teamwork habits and coding behaviors.

4. Methods

4.1. S1. Automatically classifying commit messages

Since our focus in this work is on the students' teamwork, we focused our analysis on the commit messages of student pairs in Projects 2 and 3, where students worked in pairs. We partitioned student commit actions into seven groups based upon their impact on the project. The categories were Implementation (*I*), Writing test cases (*T*), Bug fixing (*B*), Style fixing (*S*), Documentation (*D*), Merge (*M*), and Other (*O*). We randomly selected and manually tagged 1198 commit messages (approximately 1% of the total) from our dataset classifying them into these seven categories representing the grading rubric. The tagging was done by a graduate student who had acted as a TA for this course multiple times and was familiar with the structure of the projects. The distribution of these commit types among the manually tagged commits and one example of each category are shown in Table 2.

As part of the course, the students were taught about pair programming and were specifically asked to mention it in their commit messages after 2016. We searched for relevant keywords (i.e. "pair" as well as the whole word "pp" as some students abbreviated it to identify pair programming commits) to find commit messages mentioning pair programming. We were able

Table 2: The distribution and an example of different commit types among manually tagged data.

Commit Type	Count	Example
Implementation	433	Added Constructors for inner classes
Test Cases	209	More test cases
Bug Fixes	323	Fixed logout
Documentation	42	Added Javadoc to the class
Style	57	Fixing PMD errors
Merge	44	Merge branch 'master' of ...
Other	90	asdf

to identify a total of 322 commits among all the student commits mentioning pair programming, 220 from Project 2, and 102 from Project 3.

We then used a cascade model to classify the commit messages as shown in Figure 1. Some of these categories were easily identified by specific keywords. For example, merge commit messages are often auto-generated and always have the word "merge" in them, documentation commits often mention specific keywords such as "document" or "Javadoc", style-related commits often mention the relevant static analysis tools used in class like "PMD" or "CheckStyle", and a majority of commits that do not match any of our categories and belong in the "Others" do not include any meaningful English words, which can be easily detected by checking the commit message against English word corpus.

We first removed English stop-words and lemmatized the text in the commit messages and split camel-case words to identify trends in Java file names such as "Model1Test.java". We also added course-specific keywords to the acceptable English corpus, such as BBTP (black box test plan) or TS tests (teaching staff tests). To reduce the noise in our data and increase the accuracy of our models, we used static keyword matches to label *merge*, *documentation*, *style*, and *other* category commits which had fewer examples compared to the other categories and only kept the commits that were not tagged as these categories for being tagged as other types of commits. For the remaining tags (i.e. Implementation, Test cases, Bugfix), we used a bag of words approach and defined TF-IDF features with a maximum of 300 features and an N-gram range of 1 to 10 for each commit based upon all of the commit messages in the data. Following that, we trained binary classifiers (supervised) based on the manually labeled subset of the commit messages and used those models to predict the labels for the remaining dataset. After completing this process however, we realized that a large number of the commit messages (55%) remained unlabeled and were incorrectly categorized as *Other*. This far exceeded the distribution of *Other* commits in our randomly selected labeled subset (7%). In order to address this, we decided to use self-supervised semi-supervised learning to extend this training set using a combination of manually and automatically-labeled data. More details about this approach are discussed in Section 4.2. We then used the final auto-labeled data to define features for classification of student teams.

4.2. Self-supervised semi-supervised learning

Semi-supervised models use large amounts of unlabeled data together with labeled data to develop better classifiers (Nigam et al., 2006). This approach is becoming more popular as unla-

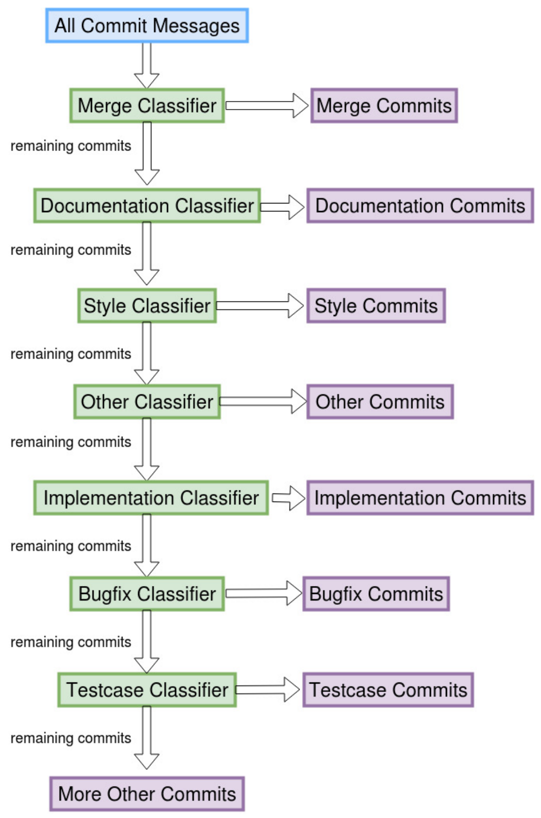


Figure 1: Cascade model for commit classification.

beled data becomes easier to collect while labeling data remains time-consuming and expensive (Nigam et al., 2006). These methods have been common for classifications in domains with large amounts of unlabeled data and smaller numbers of labeled data such as text and image classifications (Zhai et al., 2019; Miyato et al., 2016; Nigam et al., 2006). More recently, educational research has also incorporated different semi-supervised techniques for the students' performance prediction (Kostopoulos et al., 2015; Livieris et al., 2018; Livieris et al., 2019), but they have not previously been applied in analyzing students' teamwork.

Several approaches have been suggested for improving the classification models using unlabeled data such as Expectation-Maximization (EM), self-training, co-training, and graph-based methods (Nigam et al., 2006). Each of these methods fits a different kind of classification problem (Zhu, 2005). For example, if features naturally split into two sets co-training would be a good match. Since we are using a complex cascade model and our labels are not limited to two groups we selected self-training which is a more general and effective method for complex models (Zhu, 2005). In self-training (Nigam et al., 2006), we first train a classifier using labeled data. This classifier is then used to generate labels for the remaining unlabeled data. We then use both sets of data to re-train the classifier, with this process being repeated over several iterations to get better classifiers. At each iteration, we test the classifier against a held-out validation set to make sure quality is maintained going forward.

Our goal in using self-training was to increase the amount of labeled data after classification. Since our models work in a cascade iteration mode and only label positive samples, having a high precision was our priority to ensure labeled data included in the next iteration has high

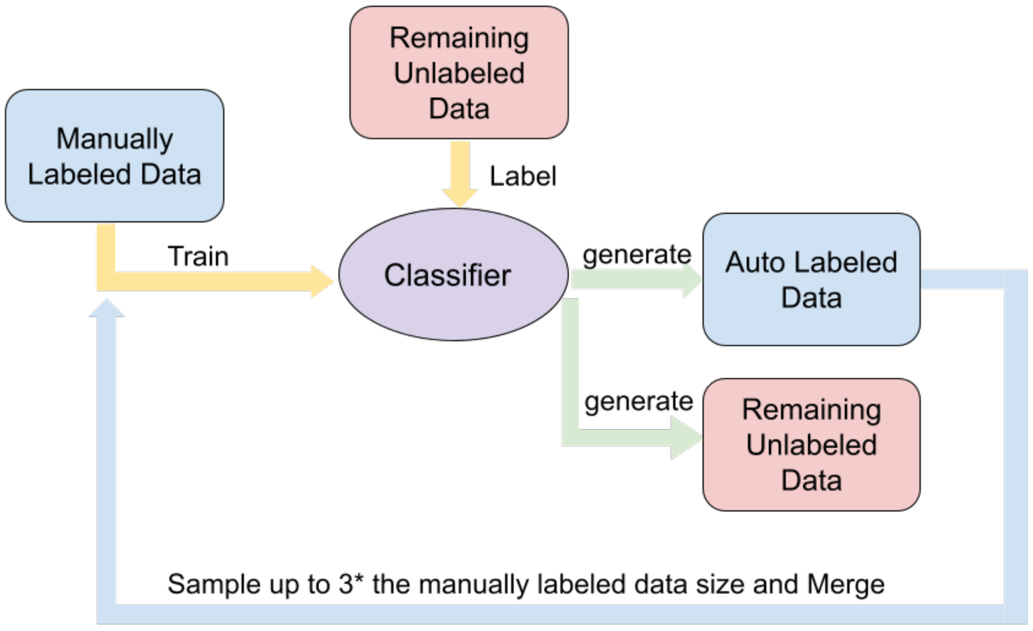


Figure 2: An overview of the self-supervised model used in this work.

confidence. For example, if a commit is selected as implementation, we want to have high confidence that it actually is an implementation commit. As long as we keep labeling correct subsets as positive and keep increasing the labeled set, leaving some of the validation set unlabeled at each iteration is not concerning. The models evolve on each iteration and it is likely that the similar values to those missed in the validation set were labeled in a different iteration. As a result, we may observe a slight drop in our model's performance, but as long as the precision remains consistently high, we can consider the selected tags to be reliable. Additionally, since the general performance of the models was not always the highest in the later iterations, we decided to keep the newly labeled samples of each iteration and remove them from future unlabeled data instead of using the final model to label all the data. This approach allowed us to ensure that lower values of recall in the models did not affect the overall labels and as long as our models were confident about a data-point at some iteration, they are labeled correctly. Since the auto-labeled data at each iteration was much larger than the manually labeled data, we decided to randomly select a subset from different labels in each iteration to not reduce the effect of expert-labeled data. The selection size was a function of the original training-set size. By trying different numbers, we found that using auto-labeled data up to three times the size of expert-labeled data reach the optimal modeling performance. An overview of the model used for commit classifications is shown in Figure 2.

We compared a series of popular text classifiers to identify the best performing model among them. The models used were: Rocchio (Joachims, 1996), Naive Bayes (NB) (Rish et al., 2001), K-Nearest Neighbors (KNN) (Manocha and Girolami, 2007), Logistic Regression (LR) (Van Houwelingen et al., 1988), Stochastic Gradient Descent (SGD) (Gardner, 1984), and Logistic Regression with Bagging (Breiman, 1996). We compared their performance with a baseline model (the best performing model among them at the first iteration, not including auto-labeled data). While trying different machine learning methods we noticed that different models labeled different amounts of data in each self-supervised iteration and some models labeled data in fewer iterations than others. Because of this difference and the variance in model performances across iterations, we needed an approach to calculate the overall performance of these models and compare them across machine learning methods.

Typically, a validation set is used for calculating the performance of self-supervised models, assuming that because of the random selection of the labeled subset, the models would perform similarly on the larger unlabeled data. We used a similar approach, by selecting a validation subset through 5-fold cross-validation in each iteration to calculate the performance of the model in that iteration. Then, we used all the training data to train a classifier and label the remaining unlabeled data. To calculate the overall performance of the models over several iterations, we took the following approach.

First, assume we have model M_k at iteration k, which has a confusion matrix (CM_k) including True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values. We can calculate the rates of each confusion matrix cell as follows, where TPR, FPR, TNR, and FNR show rates of TP, FP, TN, and FN consecutively:

$$TPR_k = \frac{\#TP_k}{\#ValidationSet} \qquad TNR_k = \frac{\#TN_k}{\#ValidationSet}$$

$$FPR_k = \frac{\#FP_k}{\#ValidationSet} \qquad FNR_k = \frac{\#FN_k}{\#ValidationSet}$$

Then, because of random selection of the validation set, we can assume that the models would perform similarly on the unlabeled set and thus, these rates would be similar in the auto-labeled subset. We can calculate the confusion matrix for that subset in iteration k (CF_{ak}), using the size of the labeled data at that iteration (S_k):

$$TP_{ak} = TPR_k \times S_k \qquad TN_{ak} = TNR_k \times S_k$$

$$FP_{ak} = FPR_k \times S_k \qquad FN_{ak} = FNR_k \times S_k$$

Then, if the model labels all data over n iterations, we can calculate the final confusion matrix (CM_f) as shown below:

$$TP_f = \sum_{k=0}^{n} TP_{ak} \qquad TN_f = \sum_{k=0}^{n} TN_{ak}$$

$$FP_f = \sum_{k=0}^{n} FP_{ak} \qquad FN_f = \sum_{k=0}^{n} FN_{ak}$$

Finally, we calculate the precision, recall, and F1-score of the model over n iterations:

$$Precision_f = \frac{TP_f}{TP_f + FP_f} \qquad Recall_f = \frac{TP_f}{TP_f + FN_f}$$

$$F1_f = \frac{2 \times Precision_f \times Recall_f}{Precision_f + Recall_f}$$

4.3. S2. AUTOMATICALLY CLASSIFYING STUDENT TEAMS

As mentioned in Section 1, we labeled the teamwork style of the students who both worked on similar parts of the project as *collaborative* (e.g. both doing some implementation and some testing), while the teams where members both had significant contributions but worked on separate parts of the project (e.g. implementation and test) were labeled *cooperative*. There were also teams where one student did the majority of work and the other student either did no work or made a small amount of changes. We labeled those teams *solo-submitting*.

To identify the students' teamwork style, we randomly selected 150 repositories (about 27% of all the teams) from each offering of the course and manually tagged them as collaborative, cooperative, and solo-submit. The tagging was done by two subject matter experts (SME), experienced TAs who are familiar with the course material and grading criteria, one of whom has acted as a TA for this course multiple times. First, a sample of 20 repositories was tagged by both SMEs independently and evaluated for agreement ($\kappa = 0.88$) and then the remaining repositories were tagged separately. Some of these teams were related to students who dropped the course or had more than two members submitting on them. After removing those teams, a total of 138 labeled teams remained. As mentioned before, the students were able to get feedback on their code by pushing it to GitHub and checking it with the teaching staff test cases. As a result, there are many cases where the students wanted to try different fixes and submitted many commits with small changes continuously until they could pass the tests. Thus, the SMEs were asked to focus on the amount of work done by each student in each category, rather than the number of commits. To make the tagging process more consistent, we added more specific definitions for the different teamwork styles. A team where both members contributed between 30%-70% to at-least two common parts of the project was considered to be collaborative. The teams where one member did the majority of work in some parts and the other member worked mostly on other parts were considered cooperative. If one member did the majority of work in most parts and the other member did not work as much, the team was labeled as solo-submit. The final manual labelling yielded 84 teams that were labeled *collaborative*, 34 that were labeled *Cooperative* and 20 that were *Solo-Submit*.

Identifying students' teamwork style by manual tagging requires a great deal of expert time and it is often difficult to come to an agreement among different experts. As a consequence, we focused on identifying students' teamwork style automatically using features from their GitHub submissions, as well as their prior individual performance (exam 1 and project 1) and the way they chose their team (i.e. self-selected vs. assigned by the instructor). This approach is also consistent with our long-term goal to monitor students' teamwork styles to provide adaptive scaffolding for group work. During an interview, one of the instructors suggested that she considered students with prior individual grades below 60 to be *at-risk*. Consequently, we added new binary features reflecting the team members' risk as well. Overall, we calculated the following features for each team member, sorting the team members such that the student with the fewest total lines of code committed to the project would be designated user 0, and the student with the most would be user 1. Our final set of features included:

- The **total number of commits** for each user in the whole project as well as the number of commits in each category (Implementation, Testing, Debugging, Documentation, Merge, Style, and Other). These features can show the students' contribution as the number of commits to the whole project and to the different areas.

- The **percentage of commits** for each user in the whole project and in different categories. This feature can distinguish between two commits in a team with a total number of 20 commits vs. in another team with a total of 100 commits.
- The total number of **additions**, **deletions**, **files changed**, and **amount of change** (i.e. additions + deletions) for each user in the whole project and by category. Additions and deletions in GitHub are measured by the lines of code each user changes in a specific commit.
- The **average** amount of **additions**, **deletions**, **files changed**, and **amount of change** per commit for each user in the whole project as well as each category.
- The **percentage** of each students' additions, deletions, files changes, and amount of change in the whole project as well as each category. Similar to the percentage of commits, this can normalize the amount of change for each team based on their total amount of activities.
- The total and average **length of commit messages** for each user in the whole project and each category. This feature can distinguish between the members who write details about their changes and the ones who submit quick commits with not much explanations for them.
- The total number of **pair programming commits** by each user as well as the total for the whole project. While using the total amount of pair programming is more intuitive, we believe that if all the pair programming is done on one person's computer, it might provide some information about the dynamics of their teamwork.
- Prior individual performance for each user (i.e. exam 1 and project 1 grades). Both exam 1 and project 1 occur at a similar time and before project 2 and project 3.
- Risk label ($grade < 60$). We added each student's risk label for exam 1 and project 1 as separate features, as well as one overall risk label for the team which shows whether or not any member of the team could be considered at-risk based on exam 1 or project 1.
- The team's selection method as a label "selected" which shows whether the students in this team requested to work together or if they were assigned by the instructor.

We ultimately defined 188 features which we normalized for our analysis. We used Recursive Feature Elimination (RFE) feature selection to select the most effective features and avoid over-fitting our models using the 15 top features. We then applied a similar cascade approach to the one used for commit classifications for this task. Because the manually-labeled data was imbalanced with Collaborative teams dominating the other two we opted to use over-sampling to balance our dataset. We tested different machine learning methods including Decision Tree, Logistic Regression, Decision Tree with Ada Boosting, and Decision Tree with Bagging. We did not use more complex models such as Random Forests and Naive Bayes as our preliminary experiments showed that they were sensitive to imbalanced data and thus unreliable with the Solo-submit and cooperative groups.

After the first round of classifications, depending on the model chosen, between 70 to 150 teams remained unlabeled. We then applied the same self-training method as mentioned in Section 4.2 to label the remaining data. We calculated the performance similarly for each model type and used the teamwork style labels generated at this stage for our next round of analysis. We then repeated the training process without including the previous grades as features, to develop a robust model. The new models not including prior performance of students, if found robust, are more suitable, especially for courses without previous grade data.

4.4. S3. Comparing students by team type

Having classified the teams we then turned to analyze the impact of the teamwork style and team formation on the students' class performance. To that end, we grouped the students by team style and compared the average performance of the students in these groups and then used pairwise Mann-Whitney p-values among the students' performance in these groups. The Mann-Whitney U test is a non-parametric test to check the null hypothesis that a random member of a list is greater than a random member of another list (MacFarland et al., 2016). We chose a non-parametric test since the distribution of our data is non-normal. We used the students' final course grades and final exam grades for their final performance and their Exam 1 and Project 1 grades to reflect their prior individual performance. We also compared the students' improvement through the semester using the Normalized Learning Gain formula as shown below and compared that among the different groups as well. If a student had scored 100 on exam 1, their improvement was considered 0.

$$Improvement = \frac{Final\_Exam - exam1}{100 - exam1}$$

4.5. S4. Help-seeking behaviors by team type

To compare the differences in help-seeking behaviors among Collaborative, Cooperative, and Solo-submit teams we first calculated the number of help-seeking activities on My Digital Hand (MDH) and Piazza for each student. MDH activities reflect when students join the queue for online or in-person office hours, make requests, receive help, and leave the session. Piazza activities, by contrast, reflect cases where students ask and answer questions in the online forum. For each team, we calculated the summation of two members' Piazza activity and MDH activity as the team's activity. Then, we used Kruskal- Wallis ANOVA tests to see if teams with different collaboration styles tend to have different help-seeking frequencies on Piazza, MDH, or in general. We also calculated the Pearson correlation between MDH and Piazza activities to assess whether teams that are more active on Piazza also attend office hours more frequently.

We further analyzed the team members' activities separately to determine if the member that is more active on MDH also asks more questions on Piazza. We categorized student teams by whether it is the same member being more active on MDH and Piazza, and performed a χ^2 test to see whether this categorization correlated with the three teamwork styles. Similarly, we categorized the team by two member's help-seeking tool preferences and performed a chi-square test to see if the teams' help-seeking preferences differ significantly between the Collaborative, Cooperative, and Solo-submit teams.

Table 3: Classification performance for static models.

	M	S	D	O
F1-score	1.000	0.966	0.868	0.748
Recall	1.000	0.977	0.920	1.000
Precision	1.000	0.955	0.821	0.597

5. Results and discussion

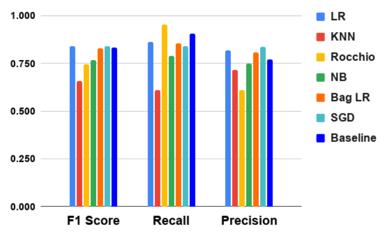
5.1. S1. Automatically classifying commit messages

Because we used static methods for classifying the Merge (M), Style (S), Documentation (D), and Other (O) commit events, the performance of these models were fixed after the first iteration of the classifications. These values are shown in Table 3. Ultimately we found that these models were successful at identifying these kinds of commits. Accurately classifying the *Other* category was more challenging due to the wide variety of commit messages associated with them, some were considered *other* because they did not fit in other categories while still others were simply unclear about what they submitted at all. Some examples are shown below.

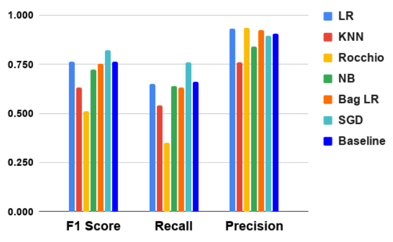
- the man was happy
- Jenkins try
- trim attempt 1
- sad
- Change in line 192/133

When classifying the Implementation (I), Testing (T), and Bugfix (B) commits, by contrast, we first compared the average performance of different models after iteration 0 and selected the best model as our supervised baseline, which in this case was the Rocchio model. This model works based on data Centroids and often works well in spaces where classes in data look like spheres. In this case, spheres and being close to a centroid would mean having or not having specific sets of words and phrases. The overall performance of different self-supervised models in classifying different labels after convergence is shown in Figures 3a, 3b, 3c. While the variation in model performance was small, our results show that the SGD model outperformed the other models in most cases (i.e. predicting one of B or T) and performed similarly to other cases (i.e. predicting I). These results also show that based on F1-score and precision, the selected self-supervised model performed better than the baseline. Not all the self-supervised models performed better or even similar to the baseline. This is in alignment with what was noted by Zhu (2005); ”if the structure of the problem and the model assumptions are not well-matched, the models might degrade in performance”. But they also noted that ”detecting such cases in advance is an open question”. It is interesting to note that the Rocchio algorithm outperformed the others on the first round and thus was selected as the baseline model and that it clearly degraded during the training process thus indicating that the model was not robust in self-training.

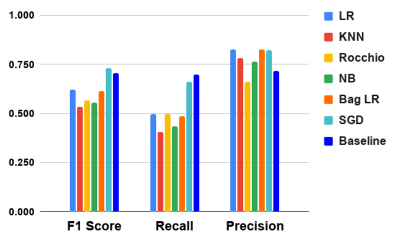
We show the number of labeled examples from each model in Figure 3d. In all the cases we observed, the amount labeled in each iteration decreases after the first two iterations but the models sometimes took different routes forward. Some models such as Naive Bayes, labeled smaller numbers in each iteration and took more iterations to converge (14 iterations for



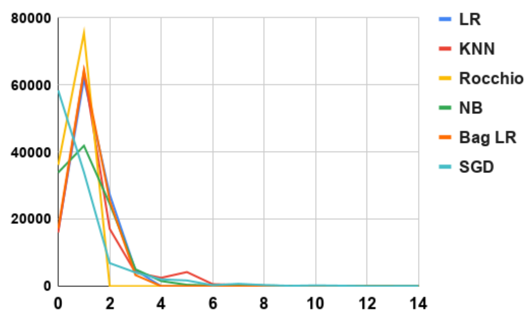
(a) Implementation prediction



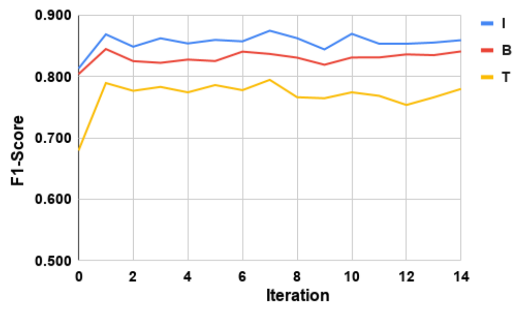
(b) Bugfix prediction



(c) Testing prediction



(d) Labeled samples in each iteration



(e) Change of SGD model performance over iterations

Figure 3: Summary of commit classifiers.

Naive Bayes), while others such as Rocchio labeled most of the data in the first two iterations. However, these trends do not seem to correlate with the model performance. As we observe, Rocchio and Logistic Regression had similar patterns in labeling data but Logistic Regression performed better over time compared to Rocchio. Considering the performance of each model over different iterations, we observe that the performance increases slightly in the first round of self-training and remains largely steady with very small changes after that. An example is shown in Figure 3e which is the change of F1-score over time for the SGD classifier. In this figure, *I* represents the performance of the classifier predicting Implementation commits, where *B* and *T* represent classifiers for Bugfix and Testing commit prediction consecutively.

The distribution of different commit messages over all the data is shown in Table 4. These distributions show us that a large portion of the students' commits belongs to implementation and fixing bugs. Having very few style-based or documentation and tests commits shows that the students often fix style issues or add documentation and tests for the projects in fewer attempts. This is mostly because they can check style errors and code coverage on their local platforms and submit once done while adding features to their code and getting a functional version of the project that passes all the teaching staff test cases is often challenging for the students and takes many attempts. As a reminder, part of the students' grade in these classes was passing the teaching staff test cases that they did not have access to. The tests were deployed on a Jenkins server connected to the students' GitHub repositories. Thus, the students needed to make the changes and submit their code to the GitHub repositories connected to Jenkins servers to check how the project was doing on those tests and whether or not the issues were fixed.

Table 4: The distribution of different commit types auto-labeled commits.

Label	Percentage in auto-labeled data
Implementation	0.39
Bugfix	0.23
Testing	0.21
Merge	0.002
Style	0.04
Documentation	0.03
Other	0.09

Table 5: Distribution of the different teamwork styles.

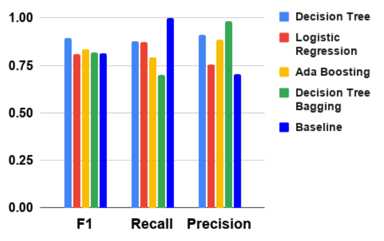
	Count	Ratio
SME tagged		
Collaborative	84	0.60
Cooperative	34	0.24
Solo-submit	23	0.16
All Data		
Collaborative	376	0.71
Cooperative	114	0.21
Solo-submit	43	0.08

5.2. S2. Automatically classifying student teams

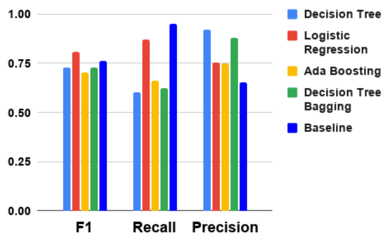
The distribution of the different teamwork styles in the tagged by expert data and all data is shown in Table 5. Similar to commit classification in Section 5.1, we first chose our baseline as the best performing model on average at the first iteration before self-training. In this case, Decision Tree with Bagging performed best and was selected. The performance comparison of different models for predicting Collaborative, Cooperative, and Solo-submitting teams are shown in Figures 4a, 4b, and 4c accordingly. As these figures show, many of the self-trained models performed as well or better than the baseline model. We observe that the recall of the models often got lower after several iterations of self-training. However, since in our cascade model we only label positive examples and leave the remaining samples for the next iteration, we believe that as long as the precision remains high (meaning the amount of false positives is low) the usability and accuracy of the labeled items will not be affected by the lower recall.

Taking a closer look at the number of labeled samples in each iteration in Figure 4d, we can see that labeling students' teamwork took fewer iterations (on average 3.5 iterations for different classifiers) compared to labeling commit messages (on average eight iterations). This difference might be due to a larger variety of students' commit message texts compared to their activities or because of the lower rate of manually labeled data to the unlabeled data in commit messages.

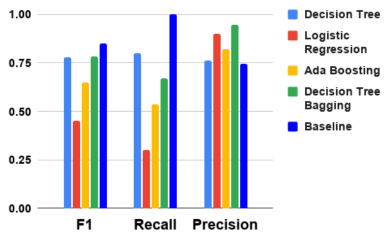
The changes in model performance that we observed over several iterations show larger fluctuations when compared to commit classification. This difference might relate to the lower general performance of these models compared to commit classifiers, which can introduce more errors to the models initially. However, if we consider the models' overall performance count-



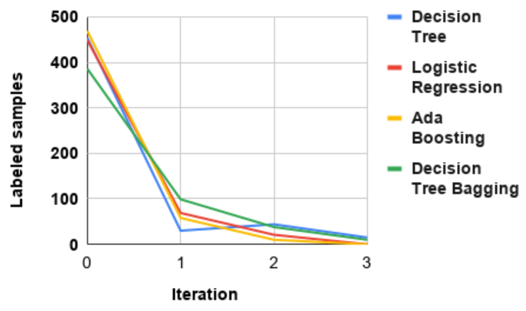
(a) Performance of models predicting collaboration



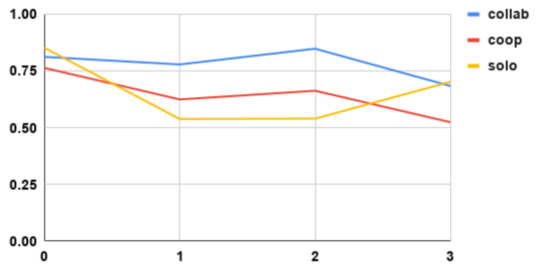
(b) Performance of models predicting cooperation



(c) Performance of models predicting solo-submission



(d) Number of classified teams in each iteration

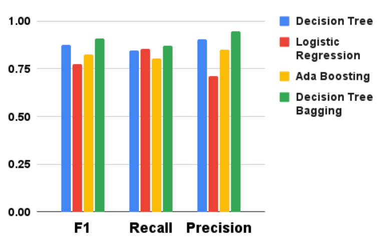


(e) Change of Decision Tree with Bagging model performance over iterations

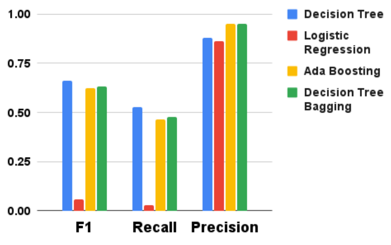
Figure 4: Summary of teamwork style classification.

ing for the number of labeled examples in each iteration (Figures 4a, 4b, 4c), we can see that in general, the selected models' performance remained close to our baseline. Thus, the degradation did not affect the overall performance of the models drastically, especially because this performance drop has been related to the models' recall in most cases. Thus, we believe that the labels generated by these models are as reliable as the labels generated by the original model, with more coverage. These models show that we can effectively identify the students' teamwork style, using automatically generated features from their commit history and their contributions to the different parts of the project. Using predictive models, the defined features, and self-supervised learning makes it easier and faster for the instructors to find the teams with harmful teamwork habits at early stages of the project and design proper interventions to enhance the students' teamwork style.

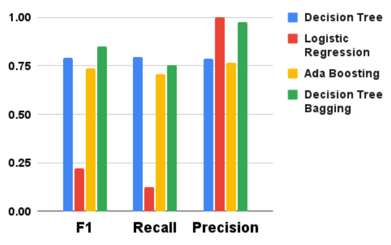
We removed the previous grade features and repeated the training process, and the performance for those resulting models are in Figures 5a, 5b, and 5c. Compared with the previous model, when predicting the collaboration or the solo-submission, the F1, Recall, or Precision score generally stays at the same level both before and after the feature removal. However, the models excluding previous grade features performed significantly worse when predicting cooperation. The F1 and recall score both went down by over 10 percent. We also witnessed a large number of false negative cases in those models, which could explain why the recall is so low. This result indicates that previous grade features are linked to forming a cooperation work style. Thus, it is necessary to include them as features when we build the classifier for teamwork style.



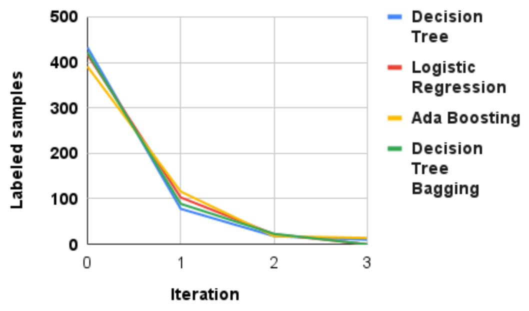
(a) Performance of models predicting collaboration



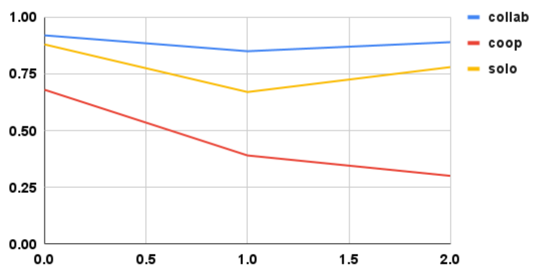
(b) Performance of models predicting cooperation



(c) Performance of models predicting solo-submission



(d) Number of classified teams in each iteration



(e) Change of Decision Tree with Bagging model performance over iterations

Figure 5: Performance of the team classification by style, excluding previous grades.

Finally, we used the auto-labeled data from the cascade model for the next stage of our analysis. The distribution of the different teamwork style groups is shown in Table 5. These numbers show that most of the students in these classes worked collaboratively and together on different parts of the project. However, on average in 15% of the projects the students divided the work and worked almost separately. In about 10% of the projects, one of the members did not have a significant contribution to the submissions. This does not necessarily mean that the inactive member did not do work, since it is possible that they exchanged their code offline and one of the members was responsible for submitting it. It still is against using GitHub in the class, since the students were supposed to learn how to use version control systems and how to share the code using them.

5.3. S3. Comparing students by team type

We compared the students' final exam grades between the three categories. The average and median grades for different groups in classes before 2020 are shown in Table 6. We focus this analysis on the data before 2020 because in that year the structure of the class changed radically due to COVID moving from in-person to remote instruction. This presented challenges for collaborative projects and led to major changes in the exams, grading, and review policies. Thus any broad comparison of performance would be inconsistent. Therefore we will first look at the overall trends in all classes before 2020 and then analyze 2020 separately.

On average the collaborating and cooperating groups had higher performance when compared to the solo-submitting groups on both the team projects and the individual exams. Mann-

Whitney tests showed that these differences were statistically significant. The Mann-Whitney p-values for pair-wise comparison of these groups are shown in Table 7 and show that the collaborating and cooperating students performed significantly better and were able to improve more in class than the solo-submitters. When we focus on the improvement scores ($\frac{FinalExam-Exam1}{100-Exam1}$) by contrast, the collaborating and cooperating teams had comparable improvement scores to the solo-submitting teams, but with much higher median values suggesting that they included a wider variation overall.

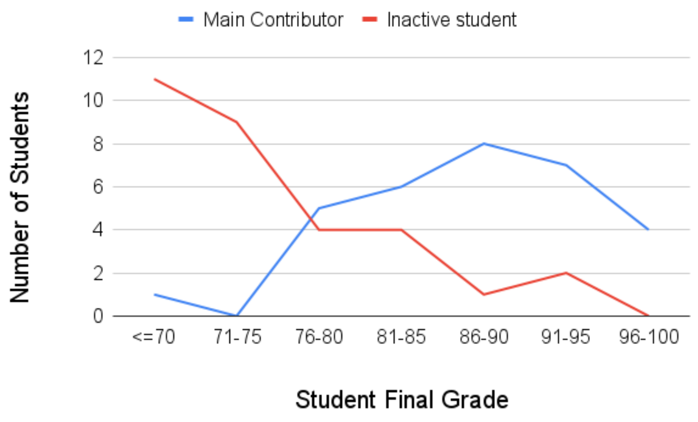
As our tables show the solo-submitters begin with lower initial performance and, as a group, have the lowest average final exam grade and the lowest median improvement, though not the lowest average compared to each of the other groups and these differences were significant. Comparing the cooperating and collaborative groups showed us that the final course grade of the collaborative group was significantly higher. However, the difference might be due to the fact that the students in the cooperative group had significantly lower grades in exam 1 and those grades were part of their final course grade. Looking at the students' grade improvements shows us that the students in the cooperating group had very similar improvement to the collaborating group and their final exam grades were also not significantly worse. Thus, it seems like there is no visible difference between the improvements of the collaborators and the cooperators, but it is clear that more of the initially better performing students chose to work collaboratively.

Table 6: The average performance of students in different categories before 2020.

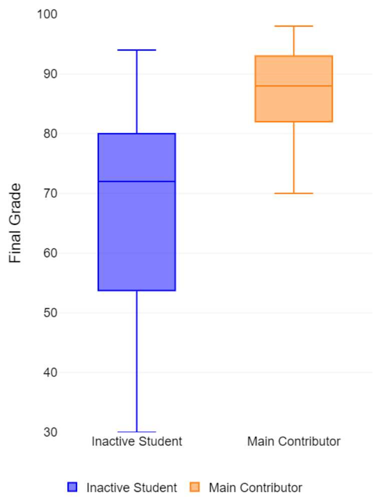
	Collaborating		Cooperating		Solo-submitting	
	Avg	Median	Avg	Median	Avg	Median
Project 1	79.58	86.54	77.19	84.82	61.30	69.26
Exam 1	80.26	82.00	77.45	79.50	72.76	71.50
Project 2	90.27	94.18	89.17	93.33	79.56	90.01
Project 3	71.20	87.76	71.94	86.92	69.03	87.73
Final Exam	88.09	93.00	86.40	92.50	75.88	86.50
Final Grade	87.65	90.00	85.92	89.00	76.84	83.00
Improve	0.34	0.56	0.31	0.59	0.31	0.37

Table 7: Mann-Whitney pair-wise p-values before 2020, values less than 0.05 are shown in bold.

	Collaborators vs. Cooperators	Cooperators vs. Solo-submitters	Collaborators vs. Solo-submitters
Project 1	0.25253	**0.00024**	**0.00001**
Exam 1	0.00736	**0.01480**	**0.00010**
Project 2	0.10502	**0.00359**	**0.00010**
Project 3	0.46206	0.48409	0.44566
Final Exam	0.24537	**0.00476**	**0.00065**
Final Grade	0.02903	**0.00158**	**0.00001**
Improve	0.49767	**0.03404**	**0.03156**



(a) Student grades within solo style teams



(b) Box plot of grade distribution

Figure 6: Student's performance of solo style teams.

Given the variation in how students performed within the solo-submitting groups, we opted to examine the performance of the students within the teams. Table 8 shows the relative performance of the students within the solo-submitting teams. As this table shows the active submitting student performed substantially better than their teammates on both Project 1 and the final course grade, and generally better on both exams. Figure 6 provides a visual comparison of the final grades. As this figure shows the inactive students were substantially worse than their active peers. These performance differences in the Project 1, Final exam and Final grade were statistically significant ($p < 0.01$) by Mann-Whitney tests. As our results also show even the active students performed worse than students in the cooperative and collaborative teams though the difference was smaller. This is consistent with our hypothesis.

These results are consistent with our hypothesis as guided by Social Learning Theory. As suggested by Social Learning Theory, the students can learn in social environments by paying attention to their peers, reproducing their work, being continuously engaged in the team process, and being motivated (Bandura and Walters, 1977). We hypothesized that a significant amount

Table 8: Performance within the solo-submitting groups.

	Solo(submitter)		Solo(non-submitter)	
	Avg	Median	Avg	Median
Project 1	73.5	75.5	55.0	59.0
Exam 1	73.59	72.00	71.93	71.00
Project 2	79.56	90.01	79.56	90.01
Project 3	69.03	87.73	69.03	87.73
Final Exam	79.50	89.00	70.90	74.00
Final Grade	87.33	88.00	66.34	72.00
Improve	0.29	0.36	0.22	0.27

Table 9: The average performance of students in different categories during Fall 2020.

	Collaborating		Cooperating		Solo-Submitter Team	
	Avg	Median	Avg	Median	Avg	Median
Project 1	86.97	92.54	86.67	91.98	77.89	92.94
Project 2	92.15	94.91	89.59	92.82	95.00	96.00
Final Grade	80.13	84.50	78.91	84.50	80.77	85.00

of contributions is needed from both team members so that they can have the conditions for Social Learning Theory and as a result, the students with more effective teamwork behaviors improve more over their courses. The results in this section support our hypothesis since not only the members of solo-submitting teams performed lower in the course, but also their individual improvements were significantly lower than each of the other two groups.

Examining the students' performance in 2020 however, shows us different trends. As the scores and comparisons in tables 10 and 11 show, there is no noticeable difference between the different types of teams. One likely explanation for this difference may be the remote nature of the class and the fact that students did not have as much of a chance to know each other and collaborate compared to when they were on campus. Additionally, the presence of the Covid-19 pandemic may have affected the students' habits and behaviors. More research is needed to address the effects of Covid-19 on student study and work habits.

Table 10: The average performance of students in different categories during Fall 2020.

	Solo (submitter)		Solo (non-submitter)	
	Avg	Median	Avg	Median
Project 1	77.89	92.94	77.89	92.94
Project 2	95.00	96.00	95.00	96.00
Final Grade	91.30	88	70.24	72.50

Table 11: Mann-Whitney pair-wise p-values during Fall 2020.

	Collaborators vs. Cooperators	Cooperators vs. Solo-submitters	Collaborators vs. Solo-submitters
Project 1	0.425	0.486	0.465
Project 2	0.473	0.183	0.148
Final Grade	0.462	0.373	0.292

We were also interested in knowing if the students are persistent in their teamwork styles or if they move from one type of engagement to another in different projects over a single semester. To analyze this, we looked into students in the years 2015-2018, which had 2 team projects. We looked into the students who transferred from one style in Project 2 to another in Project 3. The number of these transitions and their frequency across all the students starting in each style is shown in Table 12. The percentage shows the portion of each teamwork style in Project 2 transferring to another in Project 3. These numbers show that a larger portion of

Table 12: Students' change of teamwork styles from Project 2 to Project 3.

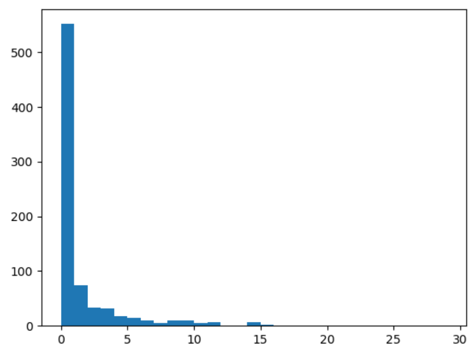
Transfer Type	# Changing Students	Proportion of Changing Students per Initial Style
Collaborative to Cooperative	30	0.05
Collaborative to Solo-submit	6	0.01
Cooperative to Collaborative	30	0.17
Cooperative to Solo-submit	2	0.01
Solo-submit to Collaborative	17	0.26
Solo-submit to Cooperative	13	0.20

students in Solo-submitting groups transfers to other styles. While the number of transfers is similar, the percentage is higher in the Solo-submitting group because of their overall smaller size. Another large portion of changes happens between Collaborative groups and Cooperative ones, with similar numbers of changes but a larger percentage of Cooperative teams changing for the next project. Overall, the students in collaborative and cooperative styles do not seem to change to the Solo-submission style. These changes can show that many students do not enjoy the solo-submission style and try to change it for the next project.

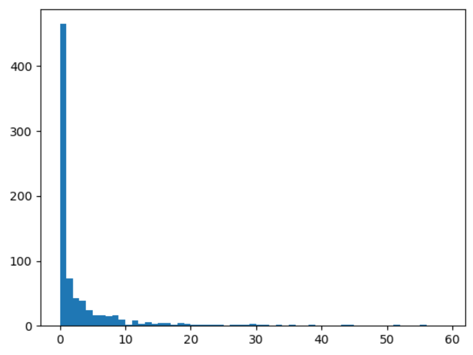
To understand the role of team selection in teamwork style, we compared the ratio of selected teams among different styles. Our observations showed that 80% of the collaborative teams, 76% of the cooperative teams, and 46% of the solo-submitting teams had selected their teammates. If we compare the ratio of collaborative, cooperative, and solo-submitting teams in selected vs. assigned teams, we observe that collaborative teams have the highest ratio in both groups. Among selected teams, 73% worked collaboratively, 21% worked cooperatively, and 5% worked in a solo-submitting style, with 1% remaining unlabeled. Among assigned teams, 58% worked collaboratively, 21% worked cooperatively, and 18% worked as solo-submitters, with 3% remaining unlabeled. Thus, our analysis showed that the solo-submitting teams have a majority of assigned team members, and while collaboration is the most common style among both selected and assigned groups, the students who are assigned to teams have a higher rate of solo-submission and a lower rate of collaboration compared to the ones who selected their teammates. This shows that the students are more likely to contribute equally to the teams they selected, compared to the teams they are assigned to.

5.4. S4.Help-seeking behaviors by team type

For each student in each team, we calculated the number of help-seeking activities in the corresponding team project. The distribution of the help-seeking activity is shown in Figure 7. For both help-seeking approaches, the majority of the students did not make any requests at all. In order to identify any correlation between the team style and the help-seeking frequency, we first calculated the summation of both members' MDH activities, the summation of their Piazza activities, and the summation of all activities. We then performed a Kruskal-Walis ANOVA (Kruskal and Wallis, 1952) test on these three measures between the three types of student teamwork styles. Corresponding p-values are 0.211, 0.403, and 0.376. We found no evidence showing that the different teamwork styles have any correlation with the members' help-seeking behavior. Also, we calculated the Pearson's correlation coefficient between the teams' total number of MDH activities and Piazza activities and observed no correlation between them (p-



(a) Total count of student team project office hours activity



(b) Total count of student team project Piazza activity

Figure 7: Distribution of help-seeking activity.

value=0.48). This observation suggests that our hypothesis about the teams more active on MDH also being more active on Piazza is incorrect.

We were also curious about the relative help-seeking behaviors within the teams. If one member is more active than another in help-seeking, is it the same person for both MDH and Piazza? For example, if students A and B are on the same team and student A makes more MDH requests, does A also make more Piazza requests? In our research, we call the teams with the same member most active on MDH and Piazza as *stay-same*, and the teams with each member most active on a different platform as *change-opposite*. In the majority of teams, at least one member had no activity on either platform. Among both active teams, we identified a total of 63 stay-same teams (16.3%) and 51 change-opposite teams (13.0%). Therefore, If one member was more active than the other on MDH, it did not necessarily mean that the member was also active on Piazza. Also, to check if this inner team help-seeking style (stay-same vs. change-opposite) has any correlation with the teamwork styles (collaborate, cooperate, and solo-submit), we performed a χ^2 test on those two categorizations. With the result p-value of 0.38, we observed no correlation between those two types of team categorization.

Finally, we investigated whether different teamwork styles have different help-seeking preferences. We categorized students by their most frequently used platform. Because the number of Piazza posts is usually more than MDH requests, we used the normalized counts to compare the activities on Piazza and MDH. Therefore, each student either prefers MDH, prefers Piazza, or has No Preference (NP). By considering both team member preferences, we categorized teams into six different types, as shown in Figure 8. We then performed a χ^2 test between them, and with a p-value of 0.127, we observed that the teamwork styles have no correlation with the team members' help-seeking preferences within teams.

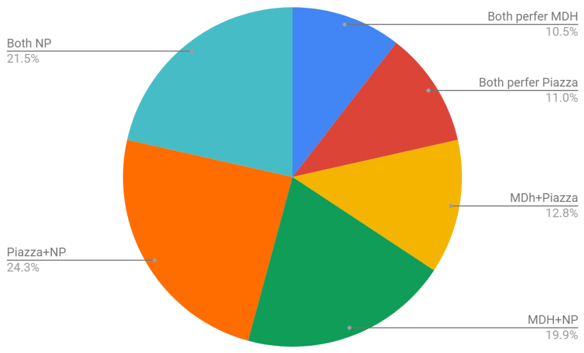


Figure 8: Member preference within each team.

6. Limitations and future work

Using the model performance on a small subset of data for estimating the model performance on unlabeled data is a common approach in evaluating semi-supervised models. This approach, while it is reliable, does build on the assumption that since the labeled subset provides a sufficient and unbiased example for the unlabeled data, the models would perform similarly. This assumption is a potential weakness of our evaluation method. Another risk of using self-supervised learning for classification tasks is the fact that the model keeps learning from the data it has already labeled. While this approach can increase the coverage and accuracy of the models, it is also possible for errors in the models to propagate across iterations. We worked to limit this propagation by limiting the use of auto-labeled data in training. Another possible approach was to only include labeled samples with high prediction confidence. However, we observed that the approach increased the chances of overfitting the models in our case.

In this work, the analysis of student teams is done solely based on their online submissions and not based on self- or peer-evaluations. Moreover, the models that we use to classify the teamwork styles rely in part on the student's prior work. We chose to focus on this data because it is what the instructors will have access to while a course is taking place. However, as a consequence of this decision, it does mean that we are relying on prior information, and it is possible that we are missing some offline interactions. This may affect our models and future findings based on them. Including coded self- and peer-evaluations and comparing them to the labeled teamwork styles may provide us with useful insights about the reliability of these models, and we may incorporate that into future work.

One area for future work will be to offer these identified patterns to the course instructors in an adaptive support platform, so they can observe different patterns and possible red flags in their classes and plan interventions if necessary. For the classifications to be useful, we will need to classify students' early submissions. Further investigation is needed to ensure that the early submissions reflect similar patterns to all submissions.

7. Conclusions

Social Learning Theory suggests that people can learn from each other if they have the motivation and the opportunity to observe and reproduce each other's work. In this study, we hypothesized that based on this theory, the students who have uneven contributions to the team projects have a lower chance of learning from each other, whether they are the ones who do most of the work or the ones that contribute less. Our results partially supported this showing that students in the solo-submitting groups prior to 2020 both performed worse than their peers in other groups with the submitting student performing better than their non-submitting peer but still gaining less. To analyze different styles in students' teamwork, we manually labeled 138 GitHub repositories of student projects in two offerings of a Java introductory course for CS majors as "Collaborative", "Cooperative", or "Solo-submit". We then used several measures based on student activities on GitHub, their prior performance, and whether they chose their teammates to automatically label all the student repositories in these classes. Finally, we compared the performance and help-seeking of the students who were members of collaborative, cooperative, and solo-submit teams and observed that the students in solo-submit teams had significantly lower final course performance, lower final exam grade, and less improvement from exam 1 to the final exam, compared to each of the other groups. These findings supported our hypothesis based on Social Learning Theory that the students in teams with uneven distribution of the work have a lower chance to learn from each other and improve in the course. We did not observe any significant difference between the help-seeking habits of students in different teamwork styles.

The number of students commit messages across these projects was high (a total of 112K), and our labeled subset was about 1% of the unlabeled set. Self-supervised learning helped us improve our models' coverage over the data and reduce the amount of commits remaining unlabeled from 50-90K (by supervised learning using different classifiers) to less than 4K while maintaining high precision in selecting the labels. Prior work has also shown very small subsets of labeled data can be effective in predicting labels of large datasets, especially in text classification where the amount of unlabeled data can get as high as millions (Yarowsky, 1995). Labeling students' teamwork patterns was also time-consuming and we had 150 labeled teams from a total of about 700. Using self-supervised learning, we increased the number of labeled teams from 380-470 (by supervised learning) to less than 10 while maintaining similar precision to supervised baselines.

The students in these classes were not graded for their amount of contributions on GitHub, nor were they directed to organize their teams in a particular way. As a result, they were able to split the work among themselves based on their choices, and what we observed here was their natural behaviors. This makes the findings in this study more likely to apply to other classes since the students' teamwork styles were not directed by the course structure. The findings of this study can be used to design adaptive support platforms for the instructors to observe a summary of students' activities and possible red flags in their behavior, such as solo-submitting. The instructors can then plan interventions in a timely manner to help students to better engage with team projects in the class. For instance, when dealing with a solo-submitting team, the instructor could intervene by asking both members why they chose not to collaborate effectively, whether the main contributor refused to distribute more work to the inactive student, whether the inactive student feels insecure about taking more responsibility, or it was just caused by bad communication? Then the instructor can explain why this work style is harmful and provide suggestions based on the situation. They could suggest the main contributor reduce their tasks or encourage the inactive student to take more responsibility; moreover, encouraging the team to work on the project together physically, like pair programming, could also help them build a more robust and healthy teamwork style. Our results also indicated that a cooperative or collaborative work style did not affect the project's success; thus, there is no need for the instructor to intervene physically with those teams.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation under grant # 1714538 "Developing Integrated Teaching Platforms to Enhance Blended Learning in STEM" PI Collin F. Lynch, Co-PIs Tiffany Barnes, and Sarah Heckman.

REFERENCES

AHMADZADEH, M., ELLIMAN, D., AND HIGGINS, C. 2005. An analysis of patterns of debugging among novice computer science students. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. ITiCSE '05. Association for Computing Machinery, 84–88.

BANDURA, A. AND WALTERS, R. H. 1977. *Social learning theory*. Vol. 1. Prentice-hall Englewood Cliffs, NJ.

BARR, T. F., DIXON, A. L., AND GASSENHEIMER, J. B. 2005. Exploring the "lone wolf" phenomenon in student teams. *Journal of Marketing Education 27,* 1, 81–90.

BLIKSTEIN, P. 2011. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*. LAK 2011. Association for Computing Machinery, 110–116.

BREIMAN, L. 1996. Bagging predictors. *Machine learning 24,* 2, 123–140.

CARTER, A. S., HUNDHAUSEN, C. D., AND ADESOPE, O. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual International Conference on Computing Education Research*. ICER'15. Association for Computing Machinery, 141–150.

CARTER, A. S., HUNDHAUSEN, C. D., AND ADESOPE, O. 2017. Blending measures of programming and social behavior into predictive models of student achievement in early computing courses. *ACM Trans. Comput. Educ. 17,* 3 (Aug.).

CHAO, P.-Y. 2016. Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education 95*, 202–215.

COMAN, I. D., ROBILLARD, P. N., SILLITTI, A., AND SUCCI, G. 2014. Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software 91*, 124–134.

EL ASRI, I., KERZAZI, N., BENHIBA, L., AND JANATI, M. 2017. From periphery to core: a temporal analysis of github contributors' collaboration network. In *Collaboration in a Data-Rich World*, L. M. Camarinha-Matos, H. Afsarmanesh, and R. Fornasiero, Eds. Springer International Publishing, 217–229.

FEICHTNER, S. B. AND DAVIS, E. A. 1984. Why some groups fail: A survey of students' experiences with learning groups. *Organizational Behavior Teaching Review 9,* 4, 58–73.

GANAPATHY, C., SHAW, E., AND KIM, J. 2011. Assessing collaborative undergraduate student wikis and svn with technology-based instrumentation: Relating participation patterns to learning. In *2011 ASEE Annual Conference & Exposition*. ASEE Conferences, 22.233.1 – 22.233.10.

GARDNER, W. A. 1984. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal Processing 6,* 2, 113–133.

GLASSY, L. 2006. Using version control to observe student software development processes. *Journal of Computing Sciences in Colleges 21,* 3, 99–106.

HECKMAN, S. AND KING, J. 2018. Developing software engineering skills using real tools for automated grading. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. Association for Computing Machinery, 794–799.

HOEGL, M. AND GEMUENDEN, H. G. 2001. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organization Science 12,* 4, 435–449.

HOSSEINI, R., VIHAVAINEN, A., AND BRUSILOVSKY, P. 2014. Exploring problem solving paths in a java programming course. In *Proceedings of the 25th Workshop of the Psychology of Programming Interest Group*, B. du Boulay and J. Good, Eds. 65–76.

IMBRIE, P., IMMEKUS, J. C., AND MALLER, S. J. 2005. Work in progress-a model to evaluate team effectiveness. In *Proceedings Frontiers in Education 35th Annual Conference*. IEEE, T4F–12.

JOACHIMS, T. 1996. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Tech. rep., Carnegie-mellon univ pittsburgh pa dept of computer science.

KAY, J., MAISONNEUVE, N., YACEF, K., AND REIMANN, P. 2006. The big five and visualisations of team work activity. In *Intelligent Tutoring Systems: 8th International Conference, ITS 2006, Jhongli, Taiwan, June 26-30, 2006. Proceedings 8*, M. Ikeda, K. D. Ashley, and T.-W. Chan, Eds. Springer, Springer Berlin Heidelberg, 197–206.

KIM, J., SHAW, E., XU, H., AND ADARSH, G. 2012. Assisting instructional assessment of undergraduate collaborative wiki and svn activities. In *Proceedings of the 5th International Conference of Educational Data Mining*, K. Yacef, O. Zaïane, A. Hershkovitz, M. Yudelson, and J. Stamper, Eds. International Educational Data Mining Society, 10–16.

KOSTOPOULOS, G., KOTSIANTIS, S., AND PINTELAS, P. 2015. Estimating student dropout in distance higher education using semi-supervised techniques. In *Proceedings of the 19th Panhellenic Conference on Informatics*, K. N. N, A. Demosthenes, and N. Mara, Eds. Association for Computing Machinery, 38–43.

KRUSKAL, W. H. AND WALLIS, W. A. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association 47,* 260, 583–621.

LEE, H.-J. 2009. Peer evaluation in blended team project-based learning; what do students find important? In *Proceedings of E-Learn 2009: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, T. Bastiaens, J. Dron, and C. Xin, Eds. Association for the Advancement of Computing in Education (AACE), 2838–2842.

LEE, H.-J., KIM, H., AND BYUN, H. 2017. Are high achievers successful in collaborative learning? an explorative study of college students' learning approaches in team project-based learning. *Innovations in Education and Teaching International 54,* 5, 418–427.

LIMA, J., TREUDE, C., FILHO, F. F., AND KULESZA, U. 2015. Assessing developer contribution with repository mining-based metrics. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 536–540.

LIN, Y.-T., WU, C.-C., HOU, T.-Y., LIN, Y.-C., YANG, F.-Y., AND CHANG, C.-H. 2016. Tracking students' cognitive processes during program debugging-an eye-movement approach. *IEEE Transactions on Education 59,* 3, 175–186.

LIU, Y., STROULIA, E., WONG, K., AND GERMAN, D. 2004. Using cvs historical information to understand how students develop software. In *Proceedings of the International Workshop on Mining Software Repositories, Edinburgh, Scotland*, A. E. Hassan, R. C. Holt, and A. Mockus, Eds. IET, 32–36.

LIVIERIS, I. E., DRAKOPOULOU, K., MIKROPOULOS, T. A., TAMPAKAS, V., AND PINTELAS, P. 2018. An ensemble-based semi-supervised approach for predicting students' performance. In *Research on e-Learning and ICT in Education: Technological, Pedagogical and Instructional Perspectives*, T. A. Mikropoulos, Ed. Springer International Publishing, 25–42.

LIVIERIS, I. E., DRAKOPOULOU, K., TAMPAKAS, V. T., MIKROPOULOS, T. A., AND PINTELAS, P. 2019. Predicting secondary school students' performance utilizing a semi-supervised learning approach. *Journal of Educational Computing Research 57,* 2, 448–470.

MACFARLAND, T. W., YATES, J. M., MACFARLAND, T. W., AND YATES, J. M. 2016. *Mann–whitney u test*. Springer, 103–132.

MAIN, J. B. AND SANCHEZ-PENA, M. 2015. Student evaluations of team members: Is there gender bias? In *2015 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–6.

MANOCHA, S. AND GIROLAMI, M. A. 2007. An empirical analysis of the probabilistic k-nearest neighbour classifier. *Pattern Recognition Letters 28,* 13, 1818–1824.

MIERLE, K., LAVEN, K., ROWEIS, S., AND WILSON, G. 2005. Mining student cvs repositories for performance indicators. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*. Association for Computing Machinery, 1–5.

MIYATO, T., DAI, A. M., AND GOODFELLOW, I. 2016. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725.*

MURPHY, C., KAISER, G., LOVELAND, K., AND HASAN, S. 2009. Retina: Helping students and instructors based on observed programming activities. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. SIGCSE'09. Association for Computing Machinery, 178–182.

NÄYKKI, P., JÄRVELÄ, S., KIRSCHNER, P. A., AND JÄRVENOJA, H. 2014. Socio-emotional conflict in collaborative learning-a process-oriented case study in a higher education context. *International Journal of Educational Research 68*, 1–14.

NGUYEN, T. AND CHUA, C. 2016. Predictive tool for software team performance. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, A. Potanin, G. Murphy, S. Reeves, and J. Dietrich, Eds. IEEE, 373–376.

NIGAM, K., MCCALLUM, A., AND MITCHELL, T. M. 2006. Semi-supervised text classification using em. In *Semi-Supervised Learning*, O. Chapelle, B. Schölkopf, and A. Zien, Eds. Adaptive Computation and Machine Learning. MIT Press, 31–51.

NORTHRUP, S. G. AND NORTHRUP, D. A. 2006. Multidisciplinary teamwork assessment: Individual contributions and interdisciplinary interaction. In *Proceedings. Frontiers in Education. 36th Annual Conference*. IEEE, 15–20.

OAKLEY, B., FELDER, R. M., BRENT, R., AND ELHAJJ, I. 2004. Turning student groups into effective teams. *Journal of Student Centered Learning 2,* 1, 9–34.

PARIZI, R. M., SPOLETINI, P., AND SINGH, A. 2018. Measuring team members' contributions in software engineering projects using git-driven technology. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–5.

PERERA, D., KAY, J., KOPRINSKA, I., YACEF, K., AND ZAÏANE, O. R. 2009. Clustering and sequential pattern mining of online collaborative learning data. *IEEE Transactions on Knowledge and Data Engineering 21,* 6, 759–772.

REID, K. L. AND WILSON, G. V. 2005. Learning by doing: Introducing version control as a way to manage student assignments. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '05. Association for Computing Machinery, 272–276.

RISH, I. ET AL. 2001. An empirical study of the naive bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*. Vol. 3. 41–46.

SALOMON, G. AND GLOBERSON, T. 1989. When teams do not function the way they ought to. *International Journal of Educational Research 13,* 1, 89–99.

SEERS, A. 1989. Team-member exchange quality: A new construct for role-making research. *Organizational Behavior and Human Decision Processes 43,* 1, 118–135.

SPACCO, J., DENNY, P., RICHARDS, B., BABCOCK, D., HOVEMEYER, D., MOSCOLA, J., AND DUVALL, R. 2015. Analyzing student work patterns using programming exercise data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE'15. Association for Computing Machinery, 18–23.

UCHIDA, S., MONDEN, A., IIDA, H., MATSUMOTO, K.-I., AND KUDO, H. 2002. A multiple-view analysis model of debugging processes. In *Proceedings International Symposium on Empirical Software Engineering*. ISESE'02. IEEE, 139–147.

VAN DEN BOSSCHE, P., GIJSELAERS, W. H., SEGERS, M., AND KIRSCHNER, P. A. 2006. Social and cognitive factors driving teamwork in collaborative learning environments: Team learning beliefs and behaviors. *Small Group Research 37,* 5, 490–521.

VAN DER DUIM, L., ANDERSSON, J., AND SINNEMA, M. 2007. Good practices for educational software engineering projects. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 698–707.

VAN GOG, T. AND KESTER, L. 2012. A test of the testing effect: acquiring problem-solving skills from worked examples. *Cognitive Science 36,* 8, 1532–1541.

VAN HOUWELINGEN, J., LE CESSIE, S., ET AL. 1988. Logistic regression, a review. *Statistica Neerlandica 42,* 4, 215–232.

VIHAVAINEN, A., LUUKKAINEN, M., AND KURHILA, J. 2013. Using students' programming behavior to predict success in an introductory mathematics course. In *Proceedings of the 6th International Conference on Educational Data Mining (EDM 2013)*, S. K. D'Mello, R. A. Calvo, and A. Olney, Eds. International Educational Data Mining Society, 300–303.

WATSON, C., LI, F. W., AND GODWIN, J. L. 2014. No tests required: comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE'14. Association for Computing Machinery, 469–474.

WEN, M., MAKI, K., DOW, S., HERBSLEB, J. D., AND ROSE, C. 2017. Supporting virtual team formation through community-wide deliberation. *Proceedings of the ACM on Human-Computer Interaction 1,* CSCW, 109.

YAROWSKY, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 189–196.

Zhai, X., Oliver, A., Kolesnikov, A., and Beyer, L. 2019. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, 1476–1485.

Zhu, X. J. 2005. Semi-supervised learning literature survey. UW Madison CS: Technical Report 1530 https://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf.