Utilizing the Constrained K-Means Algorithm and Pre-Class GitHub Contribution Statistics for Forming Student Teams

Jialin Cui
jcui9@ncsu.edu
North Carolina State University
Raleigh, USA

Fangtong Zhou
fzhou@ncsu.edu
North Carolina State University
Raleigh, USA

Chengyuan Liu
cliu32@ncsu.edu
North Carolina State University
Raleigh, USA

Qinjin Jia
qjia3@ncsu.edu
North Carolina State University
Raleigh, USA

Yang Song
songy@uncw.edu
University of North Carolina Wilmington
Wilmington, USA

Edward Gehringer
efg@ncsu.edu
North Carolina State University
Raleigh, USA

ABSTRACT

In modern software engineering education, team formation is crucial for mimicking real-world collaborative scenarios and boosting project-based learning outcomes. This paper introduces a simple, innovative, and universally adaptable method for forming student teams within a software engineering class. We utilize publicly available pre-class GitHub metrics as our input variables (e.g., number of commits, pull requests, code size, etc.). For team formation, the constrained k-means algorithm is employed. This algorithm embraces domain-specific constraints, ensuring the resulting teams not only resonate with the inherent data clusters but also meet educational requirements. Preliminary results suggest that our methodology yields teams with a harmonious blend of skills, experiences, and collaborative potentials, thereby setting the stage for enhanced project success and enriched learning experiences. Quantitative analyses show that teams formed via our approach outperform both randomly assembled teams and student self-selected teams concerning project grades. Moreover, teams created using our method also display a reduced standard deviation in grades, suggesting a more consistent performance across the board.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; • **Software and its engineering** → **Programming teams**; **Open source model**; *Object oriented development*.

KEYWORDS

Software Engineering Education, Teamwork and Collaboration, Qualitative Study, Statistical Study, GitHub

ACM Reference Format:
Jialin Cui, Fangtong Zhou, Chengyuan Liu, Qinjin Jia, Yang Song, and Edward Gehringer. 2024. Utilizing the Constrained K-Means Algorithm and Pre-Class GitHub Contribution Statistics for Forming Student Teams. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653634

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07
https://doi.org/10.1145/3649217.3653634

1 INTRODUCTION

Team-based projects have been consistently highlighted as foundational components of software engineering education. Such projects offer students a hands-on experience that closely mirrors real-world software development. Furthermore, these projects encourage collaboration, critical thinking, and problem-solving. Consequently, to prepare students effectively for their future careers, educators must incorporate applied, team project-based courses [17, 20, 21, 34, 37, 43]. The composition of software project teams is pivotal for the success of software projects. In a university setting, the right team composition significantly influences the learning experience [31]. Allowing students to form teams on their own or assigning them randomly has often led to less-than-optimal results [27, 38]. Meanwhile, there exists a widely held belief that a student's prior programming experience considerably influences their performance in software engineering courses and team projects alike. Numerous studies have explored the relationship between students' prior programming exposure and their performance in software engineering courses, consistently finding positive correlations [6, 18, 19, 24, 41, 42]. Given that students often possess varied prior programming experience, balancing project teams based on programming experience has proven beneficial, allowing less experienced members to learn from their more experienced peers [38]. A systematic mapping study [8] revealed that the majority of team formation methods (80.00%) utilize technical attributes (programming experience) to construct individual profiles before team assembly.

However, existing research has adopted diverse methods to determine students' programming experience. Some studies [26, 28, 39] rely on previous academic achievements to gauge programming experience, while others [5, 17] utilize self-developed questionnaires to ascertain this. Another study [25] designed specific exercises to assess students' technical capabilities. These varying methodologies result in two significant challenges: the inconsistency in findings, leading to difficulties in generalizing results, and challenges in replication due to the unique nature of the tools and exercises employed in prior studies.

To address these challenges, this paper presents an innovative, universally applicable method to form student teams within a software engineering course. We leverage publicly available pre-class GitHub metrics as our primary variables (e.g., number of commits, pull requests, code size) to measure students' technical ability. Various studies [10–14] have confirmed the efficacy of these metrics in quantitatively representing prior programming knowledge. For team formation, we employ the constrained k-means algorithm. This algorithm embraces domain-specific constraints, ensuring the resulting teams not only resonate with the inherent data clusters but also meet educational requirements. Preliminary results suggest that our methodology yields teams with a harmonious blend of skills, experiences, and collaborative potential, setting the stage for enhanced project success and enriched learning experiences. Quantitative analyses indicate that teams formed using our approach outperform both randomly assembled teams and student self-selected teams in terms of project grades. Moreover, teams created through our method display a reduced standard deviation in grades, suggesting not only more consistent performance across the board but also a more equitable learning environment.

2 RELATED WORK

The formulation of balanced teams for enhanced knowledge sharing and successful project completion has been extensively studied within software engineering, with distinct approaches in both industrial and academic settings.

2.1 Team Formation in Industry

Industry-focused research has often revolved around optimizing team composition based on a multitude of factors. Gilal et al. [22] investigated the dynamics of gender and personality in team assembly, suggesting nuanced implications on team synergy. Tsai et al. [40] presented a human resource approach factoring in individual productivity, which also touches upon the economic aspects of compensation. André et al. [3] offered a model for personnel allocation by evaluating diverse attributes like expertise and product knowledge. Similarly, da Silva et al. [15] identified a range of factors that influence managerial decisions in team formation, highlighting the complexity of the task at hand.

Innovative strategies have been proposed to tackle the inherent challenges of team formation, acknowledged as NP-hard by Lappas et al. [29]. For instance, Chiang and Lin [7] suggested an algorithmic strategy that aligns team formation with project constraints such as timeframes and budgets. From the technical perspective, Costa et al. [9] incorporated Genetic Algorithms with technical profiling to enhance team formation processes, while Di Penta et al. [16] demonstrated the utility of search-based methods in various aspects of project management, including team assembly.

The integration of both technical and social dynamics in team formation has been exemplified by Ahmad et al. [1], who designed a recommendation system that balances technical skills with interpersonal factors. Majumder et al. [33] further expanded on this by introducing a methodology for identifying socially cohesive teams, which also addresses workload distribution concerns.

2.2 Team Formation in Academia

Academic settings pose unique challenges for team formation. The transitory nature of academic projects and the lack of industrial parameters such as individual cost or long-term productivity necessitate distinct methodologies. The pioneering work of Henry [26] used academic performance and student preferences as early indicators for team potential, setting a foundation for subsequent research.

Later, H Hashiura et al. [25] leveraged project management exercises to derive data for a genetic algorithm, successfully forming teams with high satisfaction rates. Sim et al. [39] compared the outcomes of mixed-ability groups and self-selection processes, finding that students often favored the autonomy of choosing their team members. Løvold et al. [32] examined the effect of instructor-assigned versus self-selected teams on performance, concluding that the method of formation had little impact on the overall results.

Extending the investigation into the efficacy of team formation methods, D.Dzvonyaretal. [17] used custom criteria based on background data for team assembly, with the outcomes indicating satisfaction with the process. Presler-Marshall et al. [35] assessed teams formed through instructor intervention and student preferences, suggesting that team composition methods might not be as influential on performance as previously thought.

More recent studies have sought to refine the team formation process using algorithmic approaches. Sahin [36] introduced a model that outperformed traditional methods in project grades, while Kim [28] used historical data to predict team success with a high degree of accuracy. Akbar [2] employed a clustering approach, leveraging shared topic interests to facilitate team formation.

This body of work highlights the multifaceted nature of team formation, with each study contributing unique insights into the array of factors that can influence team dynamics and success. Our research builds on these foundations by proposing an accessible, data-driven approach using GitHub metrics, aiming to streamline the team formation process in academic settings.

3 CLASS STRUCTURE

Our research was carried out within a master's-level software engineering course emphasizing object-oriented design via Ruby on Rails. For the past 16 years, the same educator has supervised the course. Within this environment, students delve into the Ruby programming language and undertake projects utilizing the Ruby on Rails framework. Initial lectures address foundational concepts, including crafting use cases, refactoring, Test-Driven Design, Behavior-Driven Design, and the Agile and Scrum methodologies. The course's latter segment dives into design principles such as SOLID and various design patterns, notably the GoF patterns.

The course spans a semester and consists of two minor programming assignments followed by three significant projects. In the first month, students are introduced to the Ruby programming language. They undertake an exercise on GitHub operations and a fundamental programming task to familiarize themselves with the properties and data structures of Ruby. In the subsequent month, the focus shifted to the Ruby on Rails framework. Students use what they've learned to develop a full-stack web application. For deliverables, they submit both their deployed applications and their corresponding repositories.

From the third month, students collaborate in teams of three, working on a variety of Open Source Software (OSS) projects, all of which are hosted on GitHub. Although various OSS projects have been integrated into the curriculum, the majority originate

from an NSF-funded project led by the course instructor. Project themes encompass refactoring, test writing, and addressing sets of related bugs within existing codebases. In the final month, students, grouped in teams of 3 or 4, embark on their concluding projects. Drawing on the design knowledge they've learnt throughout the course, they either enhance the current system by adding features or updating significant functionalities. Each team is mandated to submit a pull request against the original repository. Projects undergo assessment based on their software engineering merits, with exemplary ones being merged into the primary repository.

Our research primarily centers on the second project, which pertains to full-stack web application development. The primary reason for this focus is that, in this project, every team works on the same topic, and the difficulty of the project remains consistent across different semesters. This consistency allows us to control variables and specifically analyze the impact of team formation. When considering the OSS projects in our class, each team tackles different topics, making it challenging to determine whether differences in team performance arise from the team's composition or the project's subject. For instance, it's not straightforward to assert that a team scoring 90 percent on a front-end project performed better than another team that secured 85 percent on a back-end project. To draw valid conclusions about team formation, we need to control for the variable of the project topic.

4 EXPERIMENT DESIGN

We carried out our team formation experiment in the Spring 2023 semester with 83 registered students. Initially, we gathered the pre-class contribution metrics of these students (details provided in section 5). The class was then randomly split into two groups: Group A with 42 students and Group B with 41 students.

For Group A, we applied the constrained k-means algorithm (as detailed in section 6), using their pre-class GitHub contribution metrics as input to cluster the 42 students into three groups of 14. Ideally, these three clusters should represent students with differing inter-cluster technical abilities but similar intra-cluster technical abilities. The instructor then formed teams of three, selecting one student from each cluster randomly to ensure each team had members from all clusters.

Group B acted as our control group. For this group, no algorithm was applied. The instructor randomly assigned these students into 13 teams of three and one team of two. This group represents teams that are formed randomly.

In the subsequent Fall 2023 semester, we had 94 registered students. We call this Group C. For Group C, we didn't intervene in the team formation process, allowing students to form teams on their own. This resulted in 30 teams of three and two teams of two. These 32 teams served as another control group, representing teams that were self-formed by students.

Subsequently, we conducted an analysis to discern the differences among teams formed using our approach, those formed randomly, and teams arising from student preferences.

5 DATA COLLECTION

5.1 GitHub ID

We collected project submissions in the form of pull requests from Fall 2011 to Spring 2023. Each project was submitted via a pull request. For every submission, we identified the corresponding GitHub accounts and matched them with individual students by cross-referencing GitHub profiles with the class roster. Our data indicates that the retrieval rate of GitHub accounts over the past decade has been steadily increasing, as shown in Figure 1.

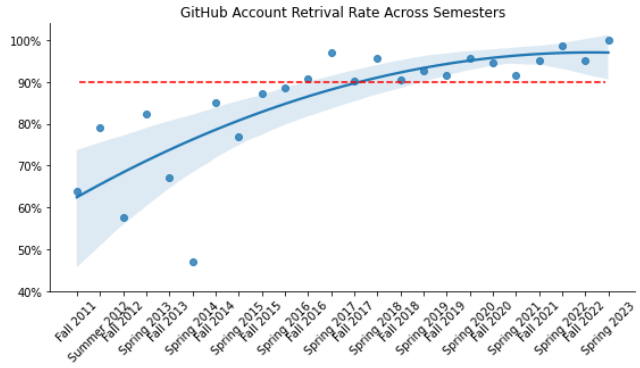

Figure 1: GitHub Account Retrieval Rate From 2011 to 2023 by Semester

Furthermore, among the collected GitHub accounts, there is a growing trend of students creating their accounts before enrolling in our class. This trend is particularly noteworthy given that our course encourages students to utilize public GitHub repositories, prompting many to establish accounts if they hadn't done so previously. The rise in students with pre-existing accounts signifies that we can gather more comprehensive data regarding their pre-course GitHub contributions. As depicted in Figure 2, there has been a significant increase in the number of students with pre-existing GitHub accounts enrolling in our courses over the past decade.

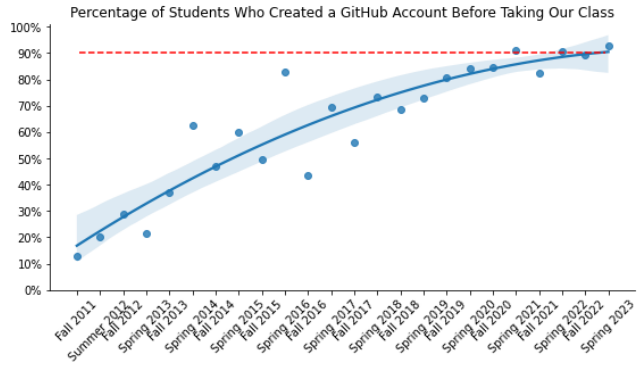

Figure 2: Proportion of Students with Pre-existing GitHub Accounts from 2011 to 2023

Given this trend, we believe that utilizing students' pre-course GitHub contribution metrics as indicators of their technical proficiency and subsequently using these metrics to form balanced teams is a feasible approach. It's worth mentioning that in recent semesters, approximately 90% of the students already possessed records prior to the start of the class. We expect this proportion to grow, reflecting the broader acceptance and use of GitHub within the community.

5.2 Pre-class GitHub Contribution Metrics

Specifically, for the Spring 2023 semester, we collected students' GitHub accounts at the beginning of the course. We then leveraged the GitHub API to create a tool that retrieves publicly available user

GitHub contribution statistics, starting from the account's inception until the beginning of the Spring 2023 semester (pre-class GitHub contribution). Table 1 outlines the metrics employed in this study. It is worth noting that several metrics aren't provided directly by GitHub; some calculations are required on our part. For example, while GitHub discloses the amount of code written by a user in a specific language for a given repository, it's our job to calculate the cumulative code volume across different languages. Furthermore, we calculated the code volume within a user's repositories for each of the top 10 languages on GitHub [23], including Python, JavaScript, Java, TypeScript, Shell, C++, Ruby, PHP, C#, and C. We also aggregated similar metrics into unified indicators; for instance, we combined all types of user comments into a singular count.

Table 1: GitHub Metrics Collected from GitHub

Metric	Explanation
Days	Days of GitHub experience.
A	The number of private contributions and public commits made by this user.
B	The number of comments made by this user in commits, issues, gists, and pull request discussions.
C	The number of PR and PR reviews made by this user.
D	The number of issues, gists, and projects created by this user.
E	The number of different languages used in all user's repositories.
F	The size of code written in GitHub popular languages in all user's repositories.
G	The total number of repositories that the user own or collaborate on.

It's important to note that all contributions logged pertain exclusively to those made by the user prior to the semester's commencement. This serves as a logistical representation of a student's technical proficiency at the beginning of the course.

6 METHODOLOGY

6.1 Constrained k-mean for Team Formation

The k-means clustering algorithm is a widely-used method that partitions a dataset into k distinct, non-overlapping clusters based on their distances to the centroid of these clusters. Given a set of observations $(x_1, x_2, \ldots, x_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k ($k \leq n$) sets $S = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS). Formally, the objective is:

$$\arg\min_{\mathrm{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mu_i\|^2$$

where μ_i is the mean of points in S_i.

However, the vanilla k-means algorithm occasionally encounters issues like empty clusters or clusters with very few points, leading to suboptimal or non-intuitive results. The constrained k-means algorithm [4] is an enhancement to the traditional approach, which tactically integrates constraints to preempt these pitfalls. Specifically, it can be tailored to ensure that each cluster houses at least a predefined minimum number of data points. This is achieved by incorporating a cluster assignment step fortified with constraints. These constraints can be efficiently tackled using methods like linear programming or network simplex, thereby ensuring a substantial population density within every formed cluster. This constrained approach not only provides more interpretable and meaningful clusters but also tends to produce more robust results in practice.

In our scenario, using the vanilla k-means for Group A would not yield the desired number of clusters, and the distribution of students across clusters would be unequal. For instance, a student with an extensive GitHub contribution record might be isolated into their own cluster. Such an outcome would be counterproductive to our team formation objectives. To address this, we employ the constrained k-means library [30], setting n_clusters to 3 and size_min to 14. This configuration results in three clusters, each containing 14 students. The instructor then selects one student from each cluster to form a team, ensuring diversity by incorporating a student from every cluster within each team. This approach ensures that each team comprises students with a variety of programming experiences, circumventing scenarios where highly experienced students are grouped together separately from their novice counterparts.

6.2 Shapiro-Wilk test, Levene's test, Student's t-test, and Mann-Whitney U-test

To compare the results of our team formation approach statistically with other prevalent methods, such as random team assignment and student self-selection, we first employed the Shapiro-Wilk test on the project grades for Group A, Group B, and Group C teams. The Shapiro-Wilk test is particularly apt for our dataset since its sizes (14 teams in Group A, 14 teams in Group B, and 32 teams in Group C) are relatively small, rendering the central limit theorem inapplicable.

The Shapiro-Wilk test is renowned for verifying dataset normality. This test assesses the null hypothesis, which posits that a sample $x_1, x_2, \ldots, x_n$ originates from a normal distribution. A significant p-value (commonly below 0.05) rejects this hypothesis, suggesting a non-normal data distribution. The test is notably effective for smaller sample sizes.

Subsequent to the normality test, we applied Levene's Test to assess variance homogeneity—a prerequisite for the Student's t-test. Levene's Test verifies if variances across multiple groups are homogeneous. A significant p-value (usually below 0.05) implies variances significantly differ across groups.

Specifically, we first applied the Shapiro-Wilk test to the project grades of Groups A, B, and C. Thereafter, we used Levene's Test on the project grades of Groups A and B, and again on Groups A and C. For datasets adhering to both the normal distribution and equal variance criteria, we employed the Student's t-test to identify statistically significant mean differences.

In instances where datasets didn't conform to the normal distribution and equal variance criteria, we opted for the Mann-Whitney U-test. This non-parametric test ascertains differences between two independent samples, especially when normality assumptions aren't met. Instead of contrasting means, it examines if the distributions of two distinct variables significantly differ.

For the Mann-Whitney U-test, the null hypothesis asserts identical group distributions, inferring a 50% probability that a data point from one group exceeds one from another group. A p-value below the common 0.05 threshold leads to this hypothesis' rejection, indicating a notable distribution difference between groups.

Within our research framework, these tests corroborate data normality, verify variance equality across groups, and highlight

significant group mean differences and dissimilarities between independent samples even when data doesn't meet parametric stipulations.

7 RESULTS

Our main results are presented in Figs 3, 4, 5 and Table 2. Fig 3 depicts the grade distribution of constrained k-means cluster formed student teams from Spring 2023. The grades are spread between 89 to 99, the distribution is more uniform across the scale with a reduced frequency of high grades compared to Figure 5. The peak frequencies are more modest, indicating a more balanced team performance overall. This could imply that the K-Means clustering method for team formation results in teams that are more evenly matched in terms of abilities and potential, leading to a more equitable distribution of grades.

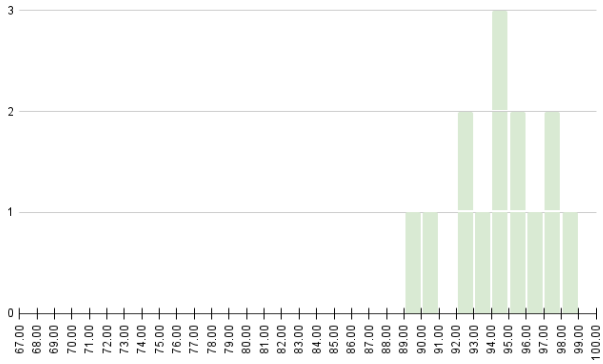

Figure 3: Grade Distribution of K-mean Cluster Student Teams, Spring 2023

Fig 4 shows the Grade Distribution of Randomly Assigned Student Teams, also from Spring 2023. In this distribution, there is a marked concentration of grades in the 90 to 95 range, but fewer teams achieve the highest grades of 95 to 100 compared to the self-selected teams (Fig 5). The distribution also has instances of lower grades not as prominent in Fig 3, suggesting a wider variance in the performance of randomly assigned teams.

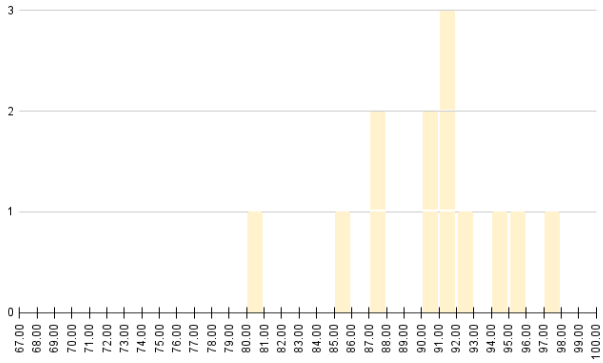

Figure 4: Grade Distribution of Randomly Assigned Student Teams, Spring 2023

Fig 5 illustrates the Grade Distribution of Self-Selected Student Teams from Fall 2023. The grades range from 67 to 100, with the distribution showing a higher concentration of teams achieving grades in the upper echelons of the scale, particularly notable is a peak around the 95 to 100 range. This suggests that self-selected teams might have a propensity to include members with similar motivations or abilities, which could lead to a polarization of outcomes where some teams perform exceptionally well. However, this also signifies a wider variance.

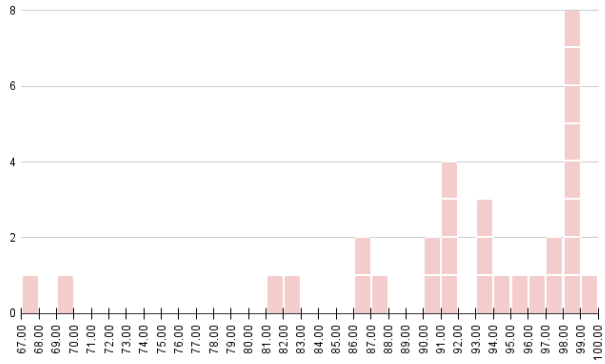

Figure 5: Grade Distribution of Self-Selected Student Teams, Fall 2023

In summary, self-selection appears to lead to a higher likelihood of exceptional performance but may also result in greater disparity. The K-Means Cluster method results in a more consistent distribution of grades, suggesting a balance in team capabilities. Random assignment shows a wider range of outcomes, which can be attributed to the unpredictable nature of random team composition.

In Table 2, we present the analysis results of team project grades. Group A represents teams we formed using the constrained k-mean clustering algorithm in Spring 2023. Group B consists of teams formed randomly during the same semester, while Group C encompasses teams that students formed on their own in Fall 2023. The row labeled "Shapiro-Wilk" displays the Shapiro-Wilk test results for the project grades across all three groups. The "Mean" row provides the average project grades, and "STDEV" indicates the standard deviation of project grades for each group. The Levene's test results present the p-value for comparisons between Group A and B, as well as Group A and C. Lastly, the Student's t-test and Mann-Whitney U-test rows show the respective p-values for comparisons between Group A and B, and Group A and C, respectively.

The cell marked in red in the first row indicates that the p-value of the Shapiro-Wilk test for Group C teams (self-formed teams) is less than 0.05, signifying that the grades are not normally distributed. The Levene's test result on the 6th row for A vs. C is 0.03, which is also less than 0.05. This suggests that the variances of these two populations are not equal. Consequently, we should use the Student's t-test to test the difference in means between Group A and B and the Mann-Whitney U-test to examine the difference in means between Group A and C.

8 DISCUSSION

8.1 Team Performance and Team Formation

The statistical analysis presented in Table 2 indicates that the team formation method employing the constrained k-mean clustering algorithm (Group A) significantly affects team performance in software engineering projects. Group A's higher mean grade (94.0) compared to Group B (90.7), supported by a Student's t-test p-value

Table 2: Students' Team Project Contribution and Grade in Fall 2023

Statistics	Group A	Group B	Group C
Shapiro-Wilk	0.80	0.95	0.00005
Mean	94.0	90.7	92.0
STDEV	2.63	5.06	8.09

Levene's test A vs B	Levene's test A vs C
0.14	0.03
Student's t-test A vs B	Mann-Whitney U-test A vs C
0.04	1.00

Notes: Group A teams: teams that we formed; Group B teams: teams that were formed randomly; Group C teams: teams that were formed by the students themselves; Shapiro-Wilk: Shapiro-Wilk test results for the project grades of Group A teams, Group B teams, and Group C teams; Mean: Mean project grades for Group A teams, Group B teams, and Group C teams; STDEV: Standard deviation of project grades for Group A teams, Group B teams, and Group C teams; Levene's test: Levene's test p-value for A vs B and A vs C; Student's t-test: Student's t-test p-value for A vs B; Mann-Whitney U-test: Mann-Whitney U-test p-value for A vs C.

of 0.04, provides evidence that algorithmically assembled teams perform better than those formed randomly. This suggests that the balanced mix of technical skills, possibly due to the inclusion of GitHub metrics in team formation, contributes to more effective collaboration and project success.

Interestingly, the comparison between Group A and student-formed teams (Group C) yields a p-value of 1.00 in the Mann-Whitney U-test, which does not indicate a statistically significant difference. This outcome challenges the assumption that algorithmic intervention is always superior to self-organization among students. It appears that the pre-existing relationships and mutual selection among peers (Group C) may compensate for the lack of systematic team balancing, potentially leading to effective collaboration due to interpersonal familiarity.

The standard deviation of the grades provides additional insight into the consistency of team performance. The notably lower standard deviation for Group A (2.63) relative to Groups B (5.06) and C (8.09) reinforces the hypothesis that a methodologically sound team assembly contributes to a more predictable and stable team performance. This stability is crucial for equitable learning experiences, as it suggests a more uniform distribution of learning opportunities and outcomes among all team members.

8.2 Broader Implications

The task of forming effective student teams in educational settings is not trivial. Educators strive to create groups that are balanced in terms of skills and experience, foster a collaborative learning environment, and enhance the overall educational outcomes. The constrained k-means clustering algorithm, as demonstrated by the research presented in this paper, emerges as a good approach to team formation that could significantly benefit pedagogical strategies across various disciplines.

The constrained k-means clustering method is advantageous for several reasons. Firstly, it incorporates domain-specific constraints that ensure teams are not only well-balanced in terms of students' technical abilities but also tailored to meet the unique requirements of each course. By utilizing pre-class metrics such as GitHub activity, educators can objectively quantify students' prior experience and programming skills, which are crucial for the success of software engineering projects. Secondly, the algorithmic nature of the constrained k-means method facilitates a fair and unbiased team assembly process. This is particularly important in educational environments where the objective is to provide equal learning opportunities to all students. Unlike self-selection, which can lead to clustering of like-skilled individuals and exacerbate the disparities in learning experiences, the constrained k-means method distributes skills more evenly across teams, which can enhance peer learning and reduce the performance gap.

Empirical evidence from our study supports the efficacy of the constrained k-means clustering approach. Teams formed using this method not only outperformed randomly assembled teams but also exhibited a higher degree of consistency in their performance, as indicated by the lower standard deviation in grades. While the performance of teams formed by the constrained k-means algorithm was not statistically different from self-selected teams, the algorithmic method provides a structured and scalable solution that can be particularly useful for large classes or online courses where personal acquaintance among students is limited.

9 THREATS TO VALIDITY

The main threats to validity arise from the formation of student teams based on their technical ability. This method might overlook other crucial factors in team dynamics, such as interpersonal skills, communication styles, and work ethics. Teams are multifaceted entities, and while balancing technical skills is crucial, other elements could influence the overall team performance and learning experience.

The study also recognizes the presence of "social factors" in student-formed teams, suggesting that pre-existing friendships or relationships could sway team performance. However, these factors haven't been quantified or extensively analyzed in the provided details, leading to potential gaps in understanding their genuine impact.

Furthermore, the differentiation between teams formed using the study's approach, random formation, and student-formed groups might be affected by external factors not addressed in the study, such as teaching methods, grading criteria, or project complexities. Deviations in these areas could introduce confounding variables that affect the results.

10 CONCLUSIONS

The paper presents a method that utilizes pre-class GitHub contributions as input for the contained k-means clustering algorithm. This generates clusters representing students' technical abilities. Instructors then form project teams by selecting students from each cluster, aiming to create teams with balanced technical capabilities. Statistical analysis confirms that teams formed using this method surpass those that are randomly formed or formed by students themselves. Moreover, teams formed by this approach have the lowest standard deviation in project scores, indicating a more equitable and balanced learning environment among teams. In conclusion, our study establishes a foundational understanding of student team formations using pre-class GitHub contribution data. In the future, we hope to better understand and improve our methods to create a better team project based learning environment.

REFERENCES

[1] Mahreen Ahmad, Wasi Haider Butt, and Abrar Ahmad. 2019. Advance Recommendation System for the Formation of More Prolific and Dynamic Software Project Teams. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. 161–165. https://doi.org/10.1109/ICSESS47205.2019.9040791

[2] Shoaib Akbar, Edward F Gehringer, and Zhewei Hu. 2018. Improving formation of student teams: a clustering approach. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 147–148.

[3] Margarita André, María G Baldoquín, and Silvia T Acuña. 2011. Formal model for assigning human resources to teams in software projects. *Information and Software Technology* 53, 3 (2011), 259–275.

[4] K.P. Bennett, P.S. Bradley, and A. Demiriz. 2000. *Constrained K-Means Clustering*. Technical Report MSR-TR-2000-65. 8 pages. https://www.microsoft.com/en-us/research/publication/constrained-k-means-clustering/

[5] Ivana Bosnić, Igor Čavrak, Marin Orlić, and Mario Žagar. 2013. Picking the right project: Assigning student teams in a GSD course. In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE.

[6] Nicholas A. Bowman, Lindsay Jarratt, K.C. Culver, and Alberto Maria Segre. 2019. How Prior Programming Experience Affects Students' Pair Programming Experiences and Outcomes. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 170–175.

[7] Hui Yi Chiang and Bertrand MT Lin. 2020. A decision model for human resource allocation in project management of software development. *IEEE Access* 8 (2020), 38073–38081.

[8] Alexandre Costa, Felipe Ramos, Mirko Perkusich, Emanuel Dantas, Ednaldo Dilorenzo, Ferdinandy Chagas, André Meireles, Danyllo Albuquerque, Luiz Silva, Hyggo Almeida, et al. 2020. Team formation in software engineering: A systematic mapping study. *Ieee Access* 8 (2020), 145687–145712.

[9] Antonio Alexandre Moura Costa, Felipe Barbosa Araújo Ramos, Mirko Barbosa Perkusich, Arthur Silva Freire, Hyggo O Almeida, and Angelo Perkusich. 2018. A Search-based Software Engineering Approach to Support Multiple Team Formation for Scrum Projects.. In *SEKE*. 474–473.

[10] Jialin Cui, Ruochi Li, Kaida Lou, Chengyuan Liu, Yunkai Xiao, Qinjin Jia, Edward Gehringer, and Runqiu Zhang. 2022. Can Pre-Class Github Contributions Predict Success by Student Teams?. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training* (Pittsburgh, Pennsylvania) *(ICSE-SEET '22)*. Association for Computing Machinery, New York, NY, USA, 40–49. https://doi.org/10.1145/3510456.3514144

[11] Jialin Cui, Runqiu Zhang, Ruochi Li, Yang Song, Fangtong Zhou, and Edward Gehringer. 2023. Correlating Students' Class Performance Based on GitHub Metrics: A Statistical Study. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 526–532.

[12] Jialin Cui, Runqiu Zhang, Ruochi Li, Fangtong Zhou, Yang Song, and Edward Gehringer. 2024. A Comparative Analysis of GitHub Contributions Before and After An OSS Based Software Engineering Class. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1*.

[13] Jialin Cui, Runqiu Zhang, Ruochi Li, Fangtong Zhou, Yang Song, and Edward Gehringer. 2024. How Pre-class Programming Experience Influences Students' Contribution to Their Team Project: A Statistical Study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (, Portland, OR, USA,) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 255–261. https://doi.org/10.1145/3626252.3630870

[14] Jialin Cui, Fangtong Zhou, Runqiu Zhang, Ruochi Li, Chengyuan Liu, and Ed Gehringer. 2023. Predicting Students' Software Engineering Class Performance with Machine Learning and Pre-Class GitHub Metrics. In *2023 IEEE Frontiers in Education Conference (FIE)*. 1–9. https://doi.org/10.1109/FIE58773.2023.10343357

[15] Fabio QB da Silva, A Cesar C Franca, Tatiana B Gouveia, Cleviton VF Monteiro, Elisa SF Cardozo, and Marcos Suassuna. 2011. An empirical study on the use of team building criteria in software projects. In *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 58–67.

[16] Massimiliano Di Penta, Mark Harman, and Giuliano Antoniol. 2011. The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. *Software: Practice and Experience* 41, 5 (2011), 495–519.

[17] Dora Dzvonyar, Lukas Alperowitz, Dominic Henze, and Bernd Bruegge. 2018. Team composition in software engineering project courses. In *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials*. 16–23.

[18] Madeline Endres, Westley Weimer, and Amir Kamil. 2021. An Analysis of Iterative and Recursive Problem Performance. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 321–327.

[19] Lidia Feklistova, Marina Lepp, and Piret Luik. 2021. Learners' Performance in a MOOC on Programming. *Education Sciences* 11, 9 (2021), 521.

[20] Norman Fenton, Shari Lawrence Pfleeger, and Robert L. Glass. 1994. Science and substance: A challenge to software engineers. *IEEE software* 11, 4 (1994), 86–95.

[21] Carlo Ghezzi and Dino Mandrioli. 2005. The Challenges of Software Engineering Education. In *Proceedings of the 2005 International Conference on Software Engineering Education in the Modern Age* (St. Louis, MO) *(ICSE'05)*. Springer-Verlag, Berlin, Heidelberg, 115–127. https://doi.org/10.1007/11949374_8

[22] Abdul Rehman Gilal, Jafreezal Jaafar, Mazni Omar, Shuib Basri, and Ahmad Waqas. 2016. A rule-based model for software development team composition: Team leader role with personality types and gender classification. *Information and Software Technology* 74 (2016), 105–113.

[23] GitHub. 2022. The top programming languages. Retrieved August 10, 2023 from https://octoverse.github.com/2022/top-programming-languages

[24] Dianne Hagan and Selby Markham. 2000. Does It Help to Have Some Programming Experience before Beginning a Computing Degree Program? *SIGCSE Bull.* 32, 3 (jul 2000), 25–28.

[25] Hiroaki Hashiura, Toru Kuwabara, Yumei Qiu, Koutarou Yamashita, Tatsuya Ishikawa, Kiyomi Shirakawa, and Seiichi Komiya. 2008. A system for supporting group exercise in software development with facilities to create an optimal plan of student grouping and team formation of each group. In *Electrical And Computer Engineering Series. Proceedings of the 7th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems. Cambridge, UK*. 149–157.

[26] Sallie Henry. 1983. A project oriented course on software engineering. *ACM SiGCSE Bulletin* 15, 1 (1983), 57–61.

[27] Sallie Henry, Nancy Miller, Wei Li, Joseph Chase, and Todd Stevens. 1999. Using software development teams in a classroom environment. In *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*. 356–357.

[28] Moon-Soo Kim. 2017. A team building algorithm based on successful record for capstone course. *Global Journal of Engineering Education* 19, 3 (2017), 243–248.

[29] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 467–476.

[30] Josh Levy-Kramer. 2023. k-means-constrained. https://pypi.org/project/k-means-constrained/ Version 0.7.3.

[31] Robert Lingard and Elizabeth Berry. 2002. Teaching teamwork skills in software engineering based on an understanding of factors affecting group performance. In *32nd Annual frontiers in Education*, Vol. 3. IEEE, S3G–S3G.

[32] Henrik Hillestad Løvold, Yngve Lindsjørn, and Viktoria Stray. 2020. Forming and assessing student teams in software engineering courses. In *Agile Processes in Software Engineering and Extreme Programming–Workshops: XP 2020 Workshops, Copenhagen, Denmark, June 8–12, 2020, Revised Selected Papers 21*. Springer.

[33] Anirban Majumder, Samik Datta, and KVM Naidu. 2012. Capacitated team formation problem on social networks. In *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining*. 1005–1013.

[34] Tom Nurkkala and Stefan Brandle. 2011. Software studio: teaching professional software engineering. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. 153–158.

[35] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. 2022. What Makes Team [s] Work? A Study of Team Characteristics in Software Engineering Projects. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 177–188.

[36] Yasar Guneri Sahin. 2011. A team building model for software engineering courses term projects. *Computers & Education* 56, 3 (2011), 916–922.

[37] Kurt Schneider, Olga Liskin, Hilko Paulsen, and Simone Kauffeld. 2015. Media, mood, and meetings: Related to project success? *ACM Transactions on Computing Education (TOCE)* 15, 4 (2015), 1–33.

[38] Thomas J Scott, Lee H Tichenor, Ralph B Bisland Jr, and James H Cross. 1994. Team dynamics in student programming projects. *ACM SIGCSE Bulletin* 26, 1 (1994), 111–115.

[39] Yan Hern Ryan Sim, Zhi Zhan Lua, Kahbelan Kalisalvam Kelaver, Jia Qi Chua, Ian Zheng Jiang Lim, Qi Cao, Sye Loong Keoh, and Li Hong Idris Lim. 2023. Experiences and Lessons Learned from Real-World Projects in Software Engineering Subject. In *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 173–177.

[40] Hsien-Tang Tsai, Herbert Moskowitz, and Lai-Hsi Lee. 2003. Human resource selection for software development projects using Taguchi's parameter design. *European Journal of operational research* 151, 1 (2003), 167–180.

[41] Susan Wiedenbeck, Deborah LaBelle, and Vennila N. R. Kain. 2004. Factors affecting course outcomes in introductory programming. In *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2004, Carlow, Ireland, April 5-7, 2004*. Psychology of Programming Interest Group, Carlow, Ireland, 11.

[42] Chris Wilcox and Albert Lionelle. 2018. Quantifying the Benefits of Prior Programming Experience in an Introductory Computer Science Course. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 80–85.

[43] Claes Wohlin and Björn Regnell. 1999. Achieving industrial relevance in software engineering education. In *Proceedings 12th conference on software engineering education and training (Cat. No. PR00131)*. IEEE, 16–25.