Evaluating the Impact of Structured Educational Resources on Git & GitHub Usage in an Introductory Programming Course

Miss Sara Osmanovic, University of Florida

Sara Osmanovic is a PhD student in Computer Science at the University of Florida under the supervision of Dr. Sara Rampazzi. She earned her B.Sc. in Computer Science from the University of Florida. Her research interests are in security of cyber-physical systems of critical infrastructure at the Cyber-Physical Systems Security (CPSec) Lab and the Florida Institute of Cybersecurity (FICS) Research.

Dr. Amanpreet Kapoor, University of Florida

Dr. Amanpreet Kapoor is an Instructional Associate Professor in the Department of Engineering Education at the University of Florida, and he teaches computing undergraduate courses in the Department of Computer and Information Science and Engineering (CISE). His research expertise is in Computing Education and Experiential Learning.

Dr. Jeremiah J Blanchard, University of Florida

Jeremiah Blanchard is an Associate Instructional Professor at the University of Florida in the Computer & Information Science & Engineering Department, where he teaches and conducts research in computer science education. He also serves as the Director of the Computer Engineering undergraduate program.

Dr. Christina Gardner-McCune, University of Florida

Christina Gardner-McCune is an Associate Professor in the Department of Computer & Information Science & Engineering at the University of Florida.

Evaluating the Impact of Structured Educational Resources on Git & GitHub Usage in an Introductory Programming Course

Abstract

Integrating version control systems (VCSs) in workflows has become pivotal in modern software development practices. Despite VCSs widespread adoption in industry, its inclusion in the computing curriculum is limited with industry professionals suggesting that computing students have challenges using such systems as they transition from academic to industry settings. Our paper explores computing undergraduate students' code collaboration and documentation practices with a VCS, Git, through source code management (SCM) platform, GitHub, in the context of a team project in a large Data Structures and Algorithms (DSA) course. We introduce a structured pedagogical intervention consisting of textual and video-based educational resources to improve students' understanding of collaboration in Git and GitHub. We further evaluate the efficacy of our intervention on students' code collaboration and documentation practices analyzing 193 code repositories with 579 students across control ($n = 324$) and intervention ($n = 255$) semesters. Our results show that most students had limited interaction with GitHub features and demonstrated weak collaboration and documentation practices when not provided with structured explicit educational resources. In addition, we found improvements after our intervention in students' collaboration and documentation practices, which were observed via common metrics such as increased commit messages per user, longer README file lengths, higher issue tracking, and more equitable teamwork distribution. Our work contributes baseline empirical knowledge on students' interaction with Git and GitHub in an introductory programming course and highlights how a structured pedagogical intervention with explicit teaching can significantly decrease the Cognitive Load of learning to use VCSs and SCMs, thus enhancing students' proficiency with these tools in a collaborative educational setting.

1 introduction

Version Control Systems are essential tools in modern software development, enabling teams to manage changes to source code over time and develop complex software [1]. Given their importance, the Computing Curricula 2023 prescribed that students should gain mastery of version control in CS degree programs [2]. Among various VCS tools, Git stands out due to its widespread adoption, robust feature set, and support for both individual and team-based projects [3]. Despite the recognized adoption of VCSs in industry and suggestions by Computing Curricula designers, many educators struggle with efficiently integrating VCSs in their classrooms and students face challenges with using VCSs, particularly when it comes to collaboration and proper documentation [4]. One of the reasons for the latter could be that most instructors of introductory programming courses would often introduce VCSs by covering basic commands, focusing on individual workflows rather than the collaborative use cases of using these tools [5]. In this regard, industry professionals have noted that recent CS graduates lack skills related to project management and software configuration management including VCSs [6]. Given these challenges, there is a pressing need to prepare our students for workflows that foster collaborative solution development. One mechanism that can prepare our students is by providing explicit structured instructional resources that offer guidance on how to utilize the various features of VCS tools like Git, and its accompa-

nying SCM platform GitHub.

Our goals for this study are to (1) understand the current practices of computing students related to code documentation and collaboration without explicit structured resources, and (2) evaluate the impact of structured resources on students' use of Git/GitHub especially pertaining to their code documentation and collaboration practices. We employ a quasi-experimental design, comparing two cohorts of undergraduate students enrolled in a DSA course over two semesters. Both cohorts were required to use GitHub for their submission, but they are not evaluated on using any particular features of Git or GitHub outside of that. This was done to ensure neither group was in any way incentivized to fluff or pad their repo statistics.

We hypothesize that structured instructions, including both textual and video-based materials, can (1) offer detailed guidance on how to utilize the various features of Git and GitHub and (2) provide students with a clearer understanding of how to collaborate on code, manage versions, and document their work comprehensively. The findings of our study have the potential to inform and enhance CS curricula, providing educators with effective strategies to integrate Git/GitHub instruction into their courses.

2 background: cognitive load theory

In psychology, the term Cognitive Load Theory (CLT) refers to the study of how instructional methods affect learning. CLT is based on the idea that humans have two types of knowledge: biologically primary and biologically secondary [7], [8]. Biologically primary knowledge is the knowledge necessary for survival. It is gained intuitively and unconsciously through our interaction with the environment and society, such as speaking our native language in terms of the mechanics of making sounds, walking, recognizing faces. On the other hand, biologically secondary knowledge is the knowledge we acquire through active and conscious learning. This includes reading, writing, specific skillsets we get taught by someone requiring active effort and mental or cognitive load. With the scheme set by these two types of knowledge, CLT studies how the biologically secondary knowledge is acquired and the cognitive load that is associated with it.

CLT defines three types of cognitive loads people have when acquiring knowledge: (1) extraneous, (2) intrinsic, and (3) germane [9], [10]. Extraneous load is the load that comes from the setting of a problem or the demands that come from the instructional design. For example, when teaching a computing concept such as traversals or searching on a graph, the instructor could either explain to the students the Breadth First Search algorithm on a problem while providing them with a solution, or they could simply present the problem at hand (searching for a vertex in a graph) and request students come up with a solution on their own. In the case where the instructor goes over a solved problem, the load of understanding how searching a graph might look is significantly lower than if students had to come up with a method of searching the graph on their own with no prior guidance or knowledge. CLT would in this case say that both approaches introduce the extraneous load, and the first method (working examples with solutions) results in a lower extraneous load [11], [12]. Extraneous load is also influenced by the sources of information and the kind of information a learner is presented with. Multiple sources of information that are required to be processed are unintelligible in isolation, and may contain irrelevant information, increasing the extraneous load [13], [14], [15].

Intrinsic load is the load that comes from the internal complexity of the problem at hand. More specifically, intrinsic load is a result of the difficulty to process information that is composed of multiple interacting components [9], [10]. In the earlier example, graph traversals require the knowledge of the graph structure, as well as the knowledge of information access and structure in computing to understand how graphs could be represented, where they exist in the space of a computer system, and how to access them. All of these individual components working together result in an intrinsic load that may or may not be high, depending on the individual learner's grasp of each interacting component. As such, it is highly individual and personal to each learner. In education, this load is attempted to be mitigated by using modular instruction, starting with the basic components and combining them as the knowledge of learner is expected to progress. It is still, however, unclear whether this decreases intrinsic load, or just decreases the extraneous load enough for the learner to then have a manageable intrinsic load [9], [16].

Lastly, CLT also includes the germane load, which is a useful difficulty for a task to inspire learning. The idea behind germane load is that reducing the extraneous load without inducing the learners to use the freed-up resources so they could reduce the intrinsic load on their own is pointless. As such, the positive or useful difficulty of a task is the load related schema acquisition or automation [10].

It is important to note that CLT is not a learning theory, but a way to explain the connection between human cognitive architecture, learning, and instructional methods. The theory has a limited scope of enactive learning scenarios where learners gain knowledge by experiencing consequences of choices they make [9]. CLT does not provide insight into the internal or self-motivation of learners, individual learning strategies of persons acquiring knowledge, nor the impact of co-constructing knowledge with peers [9], [17].

Research of the knowledge acquisition and processing has been able to draw connections between storing information and being able to use that information later for further learning through the scope of schemas [18]. The idea behind schemas is that when we learn information, we form patterns, and based on the existing patterns in our long-term memory, we extract and process new information. For instance, a learner with a strong background in biology might understand graphs by drawing an analogy to neural networks, imagining nodes as neurons and edges as synapses. In contrast, someone more experienced in computer science might interpret graphs in terms of computer networks, seeing nodes as routers and edges as data connections. In both cases, learners are leveraging their existing schemas to make sense of graphs as a data structure.

As such, CLT would suggest that reducing the extraneous load by instruction (knowledge sharing) of core components allows learners to then put more effort into forming their own schemas that reduces their random guessing in problem solving, also reducing the intrinsic load a problem might have for that student [19], [20], [21]. This theory forms the basis of our work, as we anticipate that structured resources and explicit instruction can reduce students' intrinsic and extraneous cognitive load, thereby supporting more effective learning.

3 related work

3.1 group projects in computing education

Educators have incorporated group projects in CS courses to prepare students for their careers offering environments that simulate the industrial software engineering settings [4], [5], [22]. Additionally, VCSs such as Git provide contribution transparency that allows educators to notice unequal contribution patterns such as "cowboy" programmers and "free riders" [4]. However, the grading strategies for group projects differ between instructors, with five categories identified [4], [5] as: (1) one-grade-fits-all, (2) everyone reports what they personally did, (3) group members report relative contributions of others, (4) pop quizzes on project details, and (5) cross-validating with the results of individual work. To compute individual grades for the team project used in our study, the first three techniques are used. The group first makes a single submission that counts for everyone. The students are then required to complete a survey about their own and their teammates' performances to cross-reference and update individual grades for the students who did not exert the appropriate effort based on peer feedback.

3.2 git, github, computing education, and cognitive load theory

Git is a Version Control System (VCS) designed to track changes in source code and support collaborative software development [23]. It allows developers to maintain a detailed history of project evolution, manage branching and merging, and coordinate work across distributed teams, increasing transparency and progress tracking. GitHub is a cloud-based developer platform for source code management (SCM) that builds on Git by providing a more visual-based interface that integrates version control with additional features such as issue tracking, pull requests, and social coding functionalities [24].

Despite their widespread adoption, learning Git and GitHub can impose a significant cognitive burden on novices. Students have described several challenges using Git or GitHub. These challenges include having a flawed mental model of Git-including files and folders, handling merge conflicts, lack of knowledge on using GitHub for project management, etc. [25]. Although GitHub reduces this complexity by offering easier integration through graphic interfaces and the web-based platform, the number of components and features available to the user through the GUI can still be overwhelming and daunting. This process can generate intrinsic cognitive load, since version control concepts are inherently complex, and extraneous cognitive load, when learners must memorize unfamiliar commands or resolve confusing error messages, or simply have to figure out what information among the vast amount presented to them is relevant to their problem. For beginners, these loads can exceed working memory capacity, leading to frustration, inefficient trial-and-error learning, and delayed progress in acquiring productive workflows. This results in avoidance of the platforms unless explicitly and strictly forced to use them, passing the responsibility of interacting with the VCS and SCM platforms onto someone who's more familiar with them, or simply finding workarounds (or "hacks") to do the very minimum required without understanding the tools.

To alleviate some of these issues, the computing education research (CER) community has developed tools [26] and interactive resources [27], as well as used GitHub or Git in coursework to support students in getting familiarity with VCSs. For the latter, educators have attempted to

integrate GitHub into their courses using three ways: (1) as a submission platform for assignments [28], [29], [30], [31], [32], (2) as a feedback tool through use of pull requests (PRs) [32], [33], and (3) as a platform for hosting course content [29], [30], [31], [34].

As help for learning how to use VCSs, instructors typically offer self-developed or external instructional resources to their students. However, empirical studies on how these resources influence student learning of VCSs are limited. Our work evaluates the impact of offering structured resources on students' knowledge of how to navigate and use Git and GitHub for group projects. These resources are rooted in explicit teaching or direct instruction philosophy [35], where concepts are taught in a step-by-step manner. Prior work has found this strategy to be effective for teaching in the context of general education [36]. Additionally, Linn and Clancy's work [37] in computing education also incorporated this strategy where they used structured worked examples accompanied with an expert's commentary to help students learn to design a program. They found that explicit explanations from experts can help students learn complex problem-solving skills, which is what CLT explains as schema formation. Our work tests the efficacy of explicit resources in a new context, helping students gain proficiency with Git and GitHub.

3.3 metrics for code collaboration & documentation

Metrics mined from GitHub repos have been used for a variety of purposes in computing education, such as gauging students' prior programming experience [38], measuring processes in project-based courses [26], or understanding code contributions [39]. To evaluate the efficacy of our intervention, we identified metrics that have been used in CER literature for measuring code collaboration and documentation. These metrics were mined and measured in group project GitHub repositories (repos) in the Control and Intervention semesters. Regarding *collaboration and equitable contributions*, we examined five metrics to evaluate our intervention: (1) number of contributors to a repo, (2) duration of engagement with repo [4], [40], (3) number of commits per user [4], [6], (4) number of issues [4], [30], [31], and (5) types of commit messages with emphasis on branch creation [33], [40] and PRs [32], [33], [40]. Regarding *documentation* [41], [42], [43], the following three metrics were examined: (1) length of the README file [44], [45], (2) comment ratio (ratio of comments to the total lines of code, higher the ratio, more readable the program) [41], [42], [43], [46], and (3) commit message content (which we inductively categorized as default GitHub generated message, repeated custom message, and unique custom message) [47].

4 methods

4.1 study design

Our study investigates the students' practices with a VCS, namely Git through the SCM GitHub, in the context of an undergraduate DSA course. We introduce a structured textual and video-based intervention (described in Section 4.2) to improve and increase the use of the development platform. Lastly, we assess the impact of our intervention on student collaboration and code documentation practices using a quasi-experiment based on the posttest-only control group design [48] spanning two semesters: Fall 2023 (control) and Spring 2024 (intervention). This design facilitates the evaluation of the effects of our intervention by comparing the practices of students with Git and GitHub

with or without the intervention [48]. Our study is guided by the following research question: *How did our structured instruction intervention impact the practices of undergraduate students in computing using Git and GitHub for collaboration and code documentation?*

4.2 study context

Course: Our study was conducted in a DSA course at a large public university in the US. This course is the third course in the sequence of required courses for CS undergraduate students and runs for 16 weeks in the Spring and Fall semesters. The course uses C++, and the assessments include two exams, three projects, two individual and one final group project, and ten conceptual and programming quizzes. For our study, we use the final project due to its collaborative structure. In this project, teams of three students are expected to build an application in any programming language that implements and compares two data structures or algorithms on a real-world data set with at least 100,000 data points. The project submissions are one per group and the project has an overall weight of 10% of a student's course grade split into three parts: *Project 3a proposal* (20%, provide formative feedback on a report), *Project 3b implementation* (60%, evaluated on a report, video demo, GitHub repo link, meeting requirements on DSA and data points), and *Project 3c peer evaluation* (as a group: 10% and as an individual: 10%). Teams are *required* to submit the source code as a URL to a GitHub repo (for 3b) and are *recommended* to use GitHub for code collaboration. They can choose to collaborate via any platform they feel comfortably with. Regardless of how they chose to collaborate on the project, only GitHub interaction was used as data for our experiment.

Control Group: Students from Fall 2023 are used as a control group as no changes have been introduced during that semester. These students have taken the same prerequisite courses as the intervention group, which provided a brief overview of Git/GitHub and the importance of code documentation in a 50-minute lecture in the CS1 course and assessed students' knowledge of Git/GitHub via a quiz and a lab assignment that counted toward their grade in CS1. The quiz gauged students' familiarity with git terminology and basic commands. The lab required students to use Git to clone a repo locally, add a text file to it in their IDE, and then commit those changes and push to the remote. Therefore, the assessments are focused on the use of Git/GitHub in an individual setting rather than collaborative aspects, including project management features on GitHub. Students may or may not be using Git or GitHub after this overview in CS1. Note that after the CS1 course, students are supposed to complete CS2 and Discrete Structures courses before enrolling in our DSA course. Hence, the Fall 2023 cohort can be used as a control group because the students were not provided any structured resources on how to use Git or GitHub after the brief overview in CS1 course, but were expected to navigate Git/GitHub on their own for the DSA team project.

For the control, the students were required to *submit* their code via a link to a GitHub repo, but they were in no capacity required to use any feature of Git or GitHub other than file upload. They were allowed to work on the code with local collaboration, share code via external platforms, or use Git/GitHub based on their personal preference and comfort.

Intervention: Our intervention was rolled out in Spring 2024 in the DSA course after the release of Project 3b on the Canvas learning management system and included structured instruc-

tional resources explaining how to collaborate on GitHub for project development and collaboration. These materials consisted of a GitHub template repo [49] created for the project, containing a markdown file called RESOURCES.md with textual instructions on how to use Git and GitHub. The instructions were organized to showcase all the features and tools the GitHub interface offers that would be useful in code collaboration, such as issue tracking, milestones, branching, commits, PRs, and merging in local and remote environments. It's worth noting that these resources were not a required reading material for the students, but were recommended in lectures and discussions as an aid for their project management. The resources were provided in the form of links to the template repository and the videos in the instructions page for the assignment.

In addition to the textual instructions, the resources included three videos. The first video (20 minutes) [50] showcased the GitHub features necessary for the project, like the markdown file, but in video form to accommodate diverse learners. The second video (26 minutes) [51] reviewed local development using Git and GitHub with a focus on individual contributions, and the third video (11 minutes) [52] described a role-play recording of an example project as a walkthrough of the development workflow. Each video had sub-labels for navigation to topics. The first two videos were recorded by the first author, and the last video was developed by the first author and two undergraduate teaching assistants for the course, role-playing as a project group. These materials were reviewed by the course instructor, and any discrepancies were resolved by editing or re-recording the videos. To assess the reception of the intervention in Spring 2024, the students were also asked to fill out an anonymous survey post the intervention.

Like the control semester, the students were *only* required to *submit* the code as a link to a GitHub repo for final assessment. The only language used in the instructions for the assignment that would encourage students to use Git/GitHub more were the instructional documents and videos. As such, *both the control and intervention have the opportunity, but not the requirement, of using Git and GitHub for collaboration throughout development.*

4.3 participants

The participants in this study consist of 579 undergraduate students enrolled in a DSA course offered in a large public university in the US during the Fall 2023 (control, $n = 324$) and Spring 2024 (intervention, $n = 255$) semesters who consented to research and completed a group project. The students come from diverse academic backgrounds in CS and exhibit varying programming proficiency and familiarity with Git/GitHub. The course enrollments for Fall 2023 and Spring 2024 were 546 and 439 (Response rate: 60%). We used data from 193 student groups with three members each ($N = 579$) and included in our analysis changes to the repos made before the project deadline. The groups were self-appointed when possible, and the students who didn't choose a group themselves were randomly assigned a group. For this study, we only examined groups consisting of three (3) active group members, meaning we only looked at the groups that:

1. Had three members assigned for the project;
2. Had all three group members *consent* to using the data from their repo for the research;
3. Had all three group members *actively participating* in the project.

Table 1: Metrics or dependent variables used for data analysis collected from GitHub logs for answering RQ

Metric category	ID	Metric
Collaboration	M1*	Number of contributors to a repository $(1\downarrow, 2\downarrow, 3\uparrow)$ △
	M2*	Duration of engagement with the repository (in days) ↑ ▲
	M3*	Number of commits per user (Gini index) ↓ ▲
	M4*	Number of issues ↑ ▲
	M5+	Type of commit messages (merge branch ↑, pull request ↑, standard change ↓) △
Documentation	M6*	Length of README file ↑ ▲
	M7*	Comment ratio ↑ ▲
	M8+	Commit message content (default ↓, repeated custom↓, unique custom ↑) △

*	Quantitative metrics analyzed using Descriptive & Inferential Statistics
+	Qualitative metrics analyzed using Inductive Content Analysis
▲	continuous variables
△	discrete variables
↑	predicted *increase* post-intervention
↓	predicted *decrease* post-intervention

Groups that reported one or two team members not actively participating were excluded from the analysis. Most students in our DSA course are in their sophomore year (Year 2).

4.4 data collection and analysis

The log data for engagement with VCS and SCM was collected using the Python Requests library [53], [54], which interacted with the GitHub REST API [55] to extract metrics from the final project repos student groups submitted in the control and intervention semesters. We anonymized all data before our analysis. For our analysis of the RQ, we were interested in five collaboration and three documentation-centric metrics. These metrics have been elaborated in Section 3.3 and Table 1. We report descriptive statistics of our metrics extracted quantitatively or qualitatively such as means, medians, ratios, and distributions for student practices, and compare them across control and intervention semesters.

Quantitative Analysis: Our independent variable is *time*, the semester during which the data was collected (Fall 2023, Spring 2024). Our dependent variables are the metrics described in Table 1. We conducted univariate analysis to evaluate our hypotheses (e.g. null: H_0=*The distribution of the number of GitHub issues in repos is the same across the control and intervention semester.*) and assess the relationships between our independent and dependent variables. The Shapiro-Wilk test was used to test for the normality [56]. An independent-sample t-test was used as a baseline for normally distributed continuous data, otherwise we used the non-parametric equivalent, Mann-Whitney U test [56]. The Pearson Chi-square test was used for the discrete variables [56] and all tests used $\alpha = 0.05$.

Example: One of the variables we tracked for students' collaboration practices is the number of commits each student made in the repo. To understand how evenly the students distribute work

and engage with the repo, we use the Gini coefficient, often used to analyze inequality in income distribution and resource consumption [57], and the inequality of contributions in software teams [58]. Since the data set is curated to include only groups of three, those team members who did not have any contributions in the repos were included in the data set with a commit count of 0. The Gini index was computed for each team and used to measure the inequality in the contributions of team members to the repository across each team in terms of commits. The range of Gini index varies from 0 to 1, where 0 means total equality and 1 means total inequality.

Qualitative Analysis: For the inductive content analysis (M5 and M8 in Table 1), we manually categorized data using the message contents based on keyword filtering. For example, to assess the collaboration dynamics, the commit messages (M5) were split into three types: *messages about branch merging*, *messages about PR merging*, and *messages describing the standard change to code*. These types were selected based on the important features of Git/GitHub that reflect effective collaboration patterns [32], [33], [40]. Git doesn't implicitly attach commits to branches they were made on, so we cannot be more specific in categorizing data [59]. The documentation practices were tracked through commit message content (M8) which was categorized as follows: (1) *default messages generated by GitHub* (e.g., "Add files via upload", "Create README.md", etc.), (2) *the custom repeated messages made by students* (e.g., "Final commit", etc.), and (3) *the unique messages created by students* (e.g., "Data is cleaned, working on map calculations"). No commits were observed to use PR rebasing in the control or intervention cohort.

4.5 limitations

Our study is quasi-experimental as the participants in the control and intervention semesters were not randomly added to each group. A better design would be to randomly assign students to the groups. Hence, the interpretation of results should be treated with caution as our findings are observed from correlations in data. In the future, a more careful experiment could be conducted to evaluate the efficacy of structured resources. Another limitation is the assumption that the two groups (control and intervention) were completely independent of each other. That may not be the case since students could have retaken the class in the intervention semester, now having more experience with Git/GitHub than they did in the Control semester. Our course had a drop and failure rate of 5.5% meaning that up to 32 students could have retaken the course in the intervention semester. Lastly, our data is from a small sample of students recruited from a DSA course. Students had limited exposure to GitHub in our CS1 course in both semesters. It would have been valuable to understand student practices with Git/GitHub in a survey to assess if both cohorts had a similar distribution of GitHub familiarity prior to the team project.

Additionally, there are several difficulties with gathering data from Git/GitHub. One of them is that Git doesn't track the source of the merge or if it even occurred, so we rely on commit messages describing any merge as an accurate description of the actions made [59]. On GitHub, PRs appear as non-merged even if they actually are [54]. The data during the intervention semester could have also been influenced by the students adding useless content to their README files. However, students were not incentivized in any way to write README files other than the mention of their general importance for code documentation in the resources file or curated pages. They were not graded based on the README file contents, nor were they informed that their README

Table 2: Mann-Whitney U Test Results comparing Code Documentation & Collaboration Metrics for Control (C) and Intervention (I).

		Median		Mean		SD		Test values		
		C	I	C	I	C	I	U	Z	p
M2	Duration of engagement*	2.2	7.1	5.5	10.9	10.2	10.7	6745.5	5.6	< 0.001
M3	Gini index of commits**	0.44	0.30	0.44	0.30	0.20	0.15	2798	-4.66	< 0.001
M4	No. of issues**	0	0	0.8	3.2	3.1	6.7	5726	2.95	< 0.001
M6	Length of README file*	3	21	22.1	42.2	57.4	57.5	6977	6.2	< 0.001
M7	Comment ratio**	0.18	0.14	0.24	0.2	0.19	0.19	4003	-1.52	0.127

Control repos = 108 *Too many outliers to follow normal distribution
Intervention repos = 85 **Not normal under Shapiro-Wilk test (sig. < 0.05)

files would be in any way specifically considered. As such, there is no basis to believe students intentionally padded their documentation in any way. However, to ensure this hasn't happened, a sample of five random repositories and five outliers in the data were selected for manual inspection. The contents were mostly project descriptions or proposals they submitted earlier, or scarce information such as the team members and the name of their product.

Similarly, another threat to data validity would be the coauthored commits feature. This occurred very infrequently and was handled by assigning commit ownership to the person listed on GitHub as the author. This was done because many students used a different Git username from their GitHub username, and the co-authored commits mostly seemed to refer to those two usernames being considered as two separate authors.

5 findings

5.1 students' git/github practices pre and post intervention

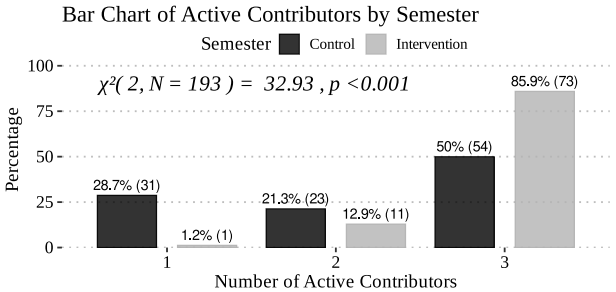

Figure 1: Active contributors to repositories across semesters.

M1. Number of contributors: The number of contributors to a repository was counted based on the commits made to that repository and split into three groups: 1-contributor, 2-contributor, and 3-contributor. Figure 1 shows how the percentages of GitHub contributions changed between semesters, and the chi-square test shows a significant dependence between the semester during which data was gathered and the distribution of contributor groups ($\chi^2(2, N = 193) = 32.93$, $p < 0.001$). Although we observe a slight decrease, the number of groups with 2 contributors to the repository did not significantly change between semesters, however, the proportion of 1-contributor and 3-contributor groups has changed significantly in favor of the three-contributor

groups, showing more cross-team engagement with Git and GitHub. The median number of contributors increased from 2.5 to 3 contributors per team. It is also important to note that in the control semester, three teams were observed to have all three people assigned to the repository with only one person contributing in terms of commits while the other 28 repositories had one person who was added to the repository. When looking at the overall number of students contributing to GitHub, these numbers total to 239 (73.8%) students in control semester and 242 (94.9%) students in intervention semester making contributions through GitHub.

M2. Duration of Engagement: The project lasted 78 days in total from the date when students were asked to select teammates to the submission of all components, where the last 34 days marked the period from which we rolled the intervention and published the development instructions for students (Project 3b). The duration of engagement was then calculated as the difference in days between the last commit and the first commit made to a given repo. In the control semester, the teams engaged with Git or GitHub on average for 5.5 days (16.2% of the development time), with a median of 2.2 days. On the other hand, the post-intervention median increased three-fold to 7.1 days, and the Mann-Whitney U test showed a significant difference between the two groups (see Table 2), with the intervention semester project teams now spending 10.9 days on average engaging with Git/GitHub (32% of the development time).

M3. Number of Commits per User: We observed an increase in the overall number of commits across semesters. In the control semester, students made a total of 1990 commits, averaging about 18.4 commits per repository and 6.1 per user. In the intervention semester, students made 3796 commits, averaging 44.7 per repository and 15.7 per user, doubling the numbers in all three aspects. To assess the equality of work distribution, we calculated the Gini index of commits in all groups and compared the results. A higher Gini index (close to 1) means higher inequality in work distribution, whereas a lower Gini index (close to 0) implies an even distribution. For the control semester, the median Gini index was 0.4 with a mode of 0.7. The intervention semester observed a decrease in the median and the mode to 0.3, leaning toward a more even work distribution. Table 2 shows that the Mann-Whitney U test run on the Gini index data confirms a significant negative difference between the two semesters.

M4. Number of Issues: GitHub issue-tracking feature proved to be an under-utilized feature in student development, with only 18.5% of groups (n=20) using issue-tracking in the control semester. After the intervention, that percentage doubled to 41.2% (n=35), and the Mann-Whitney U test (Table 2) showed a significant increase in the issue-tracking in the intervention semester. On the other hand, the median and the mode remained 0 for both semesters, with the standard deviation increasing for the intervention semester signifying a greater variation in the data.

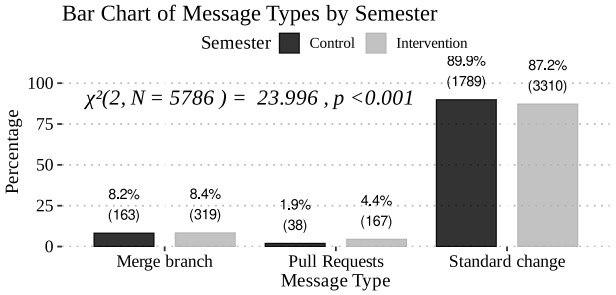

Figure 2: Commit message types across semesters.

M5. Commit Message Types: A total of 5786 commit messages were inductively coded into three categories as those pertaining to branching, PRs, or standard changes to code based on their contents. We found that students used the PR feature significantly more after the intervention, and while the use of branching also went up, the change was not significant. Figure 2 shows the results of the chi-square test run on the data, implying significant dependence of the type distribution on the semester during which data was gathered. We also observed in the data that in the control semester 47.2% (n=51) repositories had at least one commit about branching and 38.9% (n=42) repositories had at least one commit related to PRs. In the intervention semester, 69.4% (n=59) repositories had at least one commit about branching, and 32.9% (n=28) had at least one commit about PRs. Although the percentage of the groups using PRs decreased, the higher percentage of PR messages in the data suggests that those repositories engaged with them more frequently than the groups in the control semester.

M6. README File Length and Presence: The length of the README file was measured as the number of lines contained in the README.md files. The contents of the files were examined only for the outliers in the data to check for validity. In the control semester, the engagement with README files was minimal, with the median README length being 3 lines, and 41 of 108 repositories having no content or no README file. The intervention semester saw an increase in median line length to 21 lines, and a decrease in repositories without README files to only 1 out of 85. Additionally, the number of repositories with minimal content in the README file (up to 4 lines) has decreased from 20 in the control semester to 3 in the intervention semester. Both semesters saw outliers in the data with repositories containing over 100 lines, 6 in the control and 9 in the intervention semester. These repositories and 5 other randomly chosen repos were manually checked to see if students filled their README files with empty content (Lorem Ipsum, etc.). As expected, since students were not incentivized to write README files outside of the textual descriptions and video resources talking about a README file's importance for overall development quality, none of the repositories were filled with empty or placeholder content.

M7. Comment Ratio: As with the README file, we did not examine the contents of the comments, just the number of lines. Mann-Whitney U test (Table 2) showed that there wasn't a significant difference between the two semesters, so the null hypothesis couldn't be rejected. There was a slight decrease observed in the data with the median going from 0.18 to 0.14 in the intervention semester, but the data is inconclusive on the correlation between our intervention and the commenting practices of students. This decrease in comment frequency may be due to students switching mediums of documentation toward README files for instructional purposes, such as the installation and running instructions.

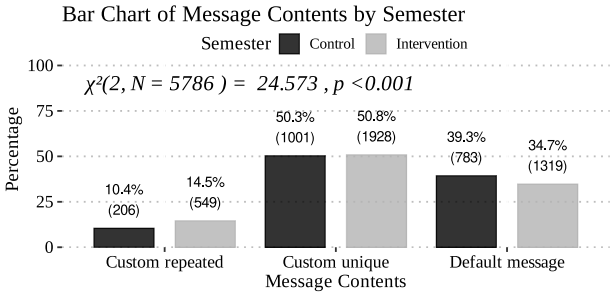

Figure 3: Commit message contents across semesters.

M8. Commit Message Contents: In terms of contents of the commit messages students made, the messages were coded into three categories based on the kind of content they had: *default messages* generated by GitHub WebUI, *custom unique messages* that appear only once in the data set, and *custom repeated messages* that appear multiple times in the set. We observed that students tend to leave the default messages generated by the GitHub WebUI unchanged when making their changes to their repo. However, during the intervention semester, they used default messages less frequently. The chi-square test run on the data (Figure 3) shows that there was a significant change in the data depending on the semester ($\chi^2(2, N = 5786) = 24.57, p < 0.001$). There was also an increase in custom repeated messages across semesters. This increase was unexpected but can be explained by students using Git/GitHub through their local IDEs, which can sometimes keep the contents of the last written commit message without warning the user (particularly Clion which is recommended in our course).

5.2 student feedback on our intervention

In this paper, we primarily use quantitative metrics from GitHub API to demonstrate the efficacy of our intervention. In the post-survey, we asked students if and how they used the resources provided to them. Out of the 165 overall responses to the survey, 127 of them answered that question, 14 of which said they did not use the resources at all. Most of those who didn't use the resources stated in their response that they were familiar with GitHub already, and a few (2) said they were not aware of the resources. It is worth noting that in the survey, we also asked them how frequently they used GitHUb prior to the class, and 3 (1.8%) out of 165 stated they have maintained open-source projects on GitHub, 7 (4.2%) worked on more than three *collaborative* projects using GitHub, and 14 (8.5%) worked on more than 3 *individual* projects using GitHub. On the other hand, 12 (7.3%) of students said they didn't even have a GitHub account prior to this project, showing the wide range of background knowledge with version control.

Qualitative data analysis from our post-survey will be described in a separate paper due to space constraints. However, we now present a few quotes from students regarding the reception of our intervention. Students described the videos and textual resources in our intervention as useful and easy to digest. For instance, S11 noted, "*The videos were most impactful. They were easy to follow and the most digestible*" and S62 stated that they "*modeled [their] repo after the template repo, and the readme helped to explain and make [them] understand how to use it and what to do*". A set of students who had more experience with GitHub described that they did not use the resources we provided. For instance, S114 stated that they "*already had experience using GitHub, so [they] didn't really need to use that many resources to begin with*". Lastly, a few students requested more beginner-friendly resources such as S31 who stated that they "*believe that the resources provided helped in all the right ways, but [they] think it could've focused more on giving beginner level advice for people who may not have ever had towork with github*."

6 discussion and conclusion

While previous work has explored students' experiences when using Git/GitHub in courses [4], [29], [30], [34], our study contributes to CER literature baseline data on students' code collaboration practices including the use of project management in GitHub by students enrolled in a DSA

course. This data includes two novel metrics – *commit message types* and *commit message contents* which we inductively coded and are underexplored in the CER literature. Additionally, we investigated how these practices changed after an intervention that explicitly provided an overview of Git and GitHub features. Our initial hypothesis that explicit instructions encourage the use of Git/GitHub was supported by the data gathered across the control and intervention semesters. Our results suggest that without explicit instructions, students tend to choose methods for collaboration they are familiar with rather than project management features afforded by GitHub or rely on those group members who are more comfortable with GitHub to make changes in their repository post-development similar to Tushev et al. [4] and Gitinabard et al. [60]. This can be explained through the Cognitive Load Theory as having reduced the extraneous load of learning Git/GitHub, allowing students to put effort toward practicing version control and building up their schemas. We show that the students are more likely to equally contribute to their repo and engage with it longer if they are given explicit instructions. In addition, their code documentation practices shift toward more varied modes of documentation, such as the README file, as opposed to relying solely on comments. Our study shows that students get more familiar with Git/GitHub when they are guided toward features of VCSs and SCMs that aid collaborative development rather than leaving it up to them to discover these features and filter out unnecessary ones. Version Control has a place in classrooms to foster transparency and structured work distribution between students and professors, but also between students themselves when required to work collaboratively.

7 recommendations for educators

We recommend Instructors follow structured instructional practices when introducing Git and GitHub to their students offering them the following: (1) multi-modal instructional resources to cater to diverse student needs; (2) themed and modularized resources to reduce cognitive load; (3) resources on challenging GitHub concepts such as handling merge conflicts, proper branching strategies, using .gitignore files, etc.; and (4) integrating GitHub beyond a submission platform across multiple courses to gradually build students' proficiency in using VCS and SCM tools so that they get acquainted with the industrial workflow for collaboration.

8 acknowledgments

The authors would like to thank Kevin Allen and Evan Hadam for their participation in the instructional videos on GitHub use, where they demonstrated key workflows through role-play scenarios that were central to this study. Their willingness to collaborate and authentic portrayals helped create realistic examples that effectively illustrate the concepts for students.

We extend our appreciation to the students in COP3530 - Data Structures and Algorithms course, who consented to share their repository activity data, which formed the foundation of our analysis.

References

[1] C. BasuMallick, *What Is Version Control? Meaning, Tools, and Advantages*, Oct. 2022. [Online]. Available: https://www.spiceworks.com/tech/devops/articles/what-is-version-control/.

[2] A. N. Kumar et al., *Computer Science Curricula 2023*. New York, NY, USA: ACM, 2024, ISBN: 9798400710339.

[3] K. Daigle, *Octoverse: the State of Open Source and Rise of AI in 2023*, Nov. 2023. [Online]. Available: https://github.blog/2023-11-08-the-state-of-open-source-and-ai/.

[4] G. W. Miroslav Tushev and A. Mahmoud, "Using GitHub in large software engineering classes. An exploratory case study," *Computer Science Education*, vol. 30, no. 2, pp. 155–186, Dec. 2020. DOI: 10.1080/08993408.2019.1696168. eprint: https://doi.org/10.1080/08993408.2019.1696168. [Online]. Available: https://doi.org/10.1080/08993408.2019.1696168.

[5] J. Hayes, T. Lethbridge, and D. Port, "Evaluating individual contribution toward group software engineering projects," in *25th International Conference on Software Engineering, 2003. Proceedings.*, Portland, Oregon, USA: IEEE Computer Society, 2003, pp. 622–627. DOI: 10.1109/ICSE.2003.1201246.

[6] A. Radermacher and G. Walia, "Gaps between industry expectations and the abilities of graduates," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '13, Denver, Colorado, USA: ACM, 2013, pp. 525–530, ISBN: 9781450318686. DOI: 10.1145/2445196.2445351. [Online]. Available: https://doi.org/10.1145/2445196.2445351.

[7] D. C. Geary, "Educating the evolved mind: Conceptual foundations for an evolutionary educational psychology," in *Psychological perspectives on contemporary educational issues*, J. S. C. J. R. Levin, Ed., IAP, 2007, pp. 1–99.

[8] D. C. Geary, "An Evolutionarily Informed Education Science," *Educational Psychologist*, vol. 43, no. 4, pp. 179–195, 2008. DOI: 10.1080/00461520802392133. eprint: https://doi.org/10.1080/00461520802392133. [Online]. Available: https://doi.org/10.1080/00461520802392133.

[9] R. Moreno and B. Park, "Cognitive Load Theory: Historical Development and Relation to Other Theories," in *Cognitive Load Theory*, J. L. Plass, R. Moreno, and R. Brünken, Eds., Cambridge University Press, 2010, pp. 9–28.

[10] J. Sweller, "Cognitive Load Theory: Recent Theoretical Advances," in *Cognitive Load Theory*, J. L. Plass, R. Moreno, and R. Brünken, Eds., Cambridge University Press, 2010, pp. 29–47.

[11] A. Renkl, "The worked-out-example principle in multimedia learning," in *The Cambridge handbook of multimedia learning*, R. E. Mayer, Ed., Cambridge University Press, 2005, pp. 229–245.

[12] F. G. Paas and J. J. Van Merriënboer, "Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach.," *Journal of educational psychology*, vol. 86, no. 1, p. 122, 1994.

[13] P. Ayres and J. Sweller, "The split-attention principle in multimedia learning," in *The Cambridge handbook of multimedia learning*, R. E. Mayer, Ed., Cambridge University Press, 2005.

[14] R. Low and J. Sweller, "The modality principle," in *The Cambridge handbook of multimedia learning*, R. E. Mayer, Ed., Cambridge University Press, 2005, pp. 147–158.

[15] J. Sweller, "The redundancy principle," in *The Cambridge handbook of multimedia learning*, R. E. Mayer, Ed., Cambridge University Press, 2005, pp. 159–167.

[16] R. Brünken, T. Seufert, and F. Paas, "Measuring Cognitive Load," in *Cognitive Load Theory*, J. L. Plass, R. Moreno, and R. Brünken, Eds., Cambridge University Press, 2010, pp. 181–202.

[17] M. Bannert, "The effects of training wheels and self-learning materials in software training," *Journal of Computer Assisted Learning*, vol. 16, no. 4, pp. 336–346, 2000.

[18] S. Kalyuga, "Schema Acquisition and Sources of Cognitive Load," in *Cognitive Load Theory*, J. L. Plass, R. Moreno, and R. Brünken, Eds., Cambridge University Press, 2010, pp. 48–64.

[19] S. Kalyuga, "Prior knowledge principle in multimedia learning," in *The Cambridge handbook of multimedia learning*, R. E. Mayer, Ed., Cambridge University Press, 2005, pp. 325–337.

[20] S. Kalyuga, "Expertise reversal effect and its implications for learner-tailored instruction," *Educational psychology review*, vol. 19, no. 4, pp. 509–539, 2007.

[21] S. Kalyuga, P. Ayres, P. Chandler, and J. Sweller, "The Expertise Reversal Effect," *Educational Psychologist*, vol. 38, no. 1, pp. 23–31, 2003. DOI: 10.1207/S15326985EP3801_4. eprint: https://doi.org/10.1207/S15326985EP3801_4. [Online]. Available: https://doi.org/10.1207/S15326985EP3801_4.

[22] L. Ohlsson and C. Johansson, "A practice driven approach to software engineering education," *IEEE Transactions on Education*, vol. 38, no. 3, pp. 291–295, 1995. DOI: 10.1109/13.406508.

[23] Git. [Online]. Available: https://git-scm.com/.

[24] *Why Choose GitHub?* [Online]. Available: https://github.com/why-github.

[25] V. Isomöttönen and M. Cochez, "Challenges and Confusions in Learning Version Control with Git," in *Information and Communication Technologies in Education, Research, and Industrial Applications*, V. Ermolayev, H. C. Mayr, M. Nikitchenko, A. Spivakovsky, and G. Zholtkevych, Eds., Cham: Springer International Publishing, 2014, pp. 178–193, ISBN: 978-3-319-13206-8.

[26] R. Glassey, "Adopting Git/Github within Teaching: A Survey of Tool Support," in *Proceedings of the ACM Conference on Global Computing Education*, ser. CompEd '19, Chengdu,Sichuan, China: ACM, 2019, pp. 143–149, ISBN: 9781450362597. DOI: 10.1145/3300115.3309518. [Online]. Available: https://doi.org/10.1145/3300115.3309518.

[27] R. T. Buxton, "Teaching Command Line and Git Skills Using Exercises with Interactive Visualizations," Ph.D. dissertation, Virginia Tech, 2023.

[28] M. Binas, "Version Control System in CS1 course: Practical experience," in *IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, High Tatras, Slovakia: IEEE, Oct. 2013, pp. 23–28, ISBN: 978-1-4799-2161-4. DOI: 10.1109/ICETA.2013.6674398.

[29] L. Haaranen and T. Lehtinen, "Teaching Git on the Side: Version Control System as a Course Platform," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15, Vilnius, Lithuania: ACM, 2015, pp. 87–92, ISBN: 9781450334402. DOI: 10.1145/2729094.2742608. [Online]. Available: https://doi.org/10.1145/2729094.2742608.

[30] J. Kelleher, "Employing git in the classroom," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, Hammamet, Tunisia: IEEE, 2014, pp. 1–4. DOI: 10.1109/WCCAIS.2014.6916568.

[31] A. Zagalsky, J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang, "The Emergence of GitHub as a Collaborative Platform for Education," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15, Vancouver, BC, Canada: ACM, 2015, pp. 1906–1917, ISBN: 9781450329224. DOI: 10.1145/2675133.2675284. [Online]. Available: https://doi.org/10.1145/2675133.2675284.

[32] C. Hsing and V. Gennarelli, "Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students and Teachers," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19, Minneapolis, MN, USA: ACM, 2019, pp. 672–678, ISBN: 9781450358903. DOI: 10.1145/3287324.3287460. [Online]. Available: https://doi.org/10.1145/3287324.3287460.

[33] Y.-C. Tu et al., "GitHub in the Classroom: Lessons Learnt," in *Proceedings of the 24th Australasian Computing Education Conference*, ser. ACE '22, Virtual Event, Australia: ACM, 2022, pp. 163–172, ISBN: 9781450396431. DOI: 10.1145/3511861.3511879. [Online]. Available: https://doi.org/10.1145/3511861.3511879.

[34] J. Feliciano, M.-A. Storey, and A. Zagalsky, "Student experiences using GitHub in software engineering courses: a case study," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16, Austin, Texas: ACM, 2016, pp. 422–431, ISBN: 9781450342056. DOI: 10.1145/2889160.2889195. [Online]. Available: https://doi.org/10.1145/2889160.2889195.

[35] B. V. Rosenshine, "Synthesis of research on explicit teaching," *Educational leadership*, vol. 43, no. 7, pp. 60–69, 1986.

[36] A. L. Archer and C. A. Hughes, *Explicit instruction: Effective and efficient teaching*. New York, United States: Guilford Publications, 2010.

[37] M. C. Linn and M. J. Clancy, "Can experts' explanations help students develop program design skills?" *International Journal of Man-Machine Studies*, vol. 36, no. 4, pp. 511–551, 1992, ISSN: 0020-7373. DOI: https://doi.org/10.1016/0020-7373(92)90095-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020737392900953.

[38] J. Cui, R. Zhang, R. Li, Y. Song, F. Zhou, and E. Gehringer, "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*, ser. ITiCSE 2023, Turku, Finland: ACM, 2023, pp. 526–532, ISBN: 9798400701382. DOI: 10.1145/3587102.3588799. [Online]. Available: https://doi.org/10.1145/3587102.3588799.

[39] J. Cui, R. Zhang, R. Li, F. Zhou, Y. Song, and E. Gehringer, "A Comparative Analysis of GitHub Contributions Before and After An OSS Based Software Engineering Class," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024, Milan, Italy: ACM, 2024, pp. 576–582, ISBN: 9798400706004. DOI: 10.1145/3649217.3653535. [Online]. Available: https://doi.org/10.1145/3649217.3653535.

[40] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in GitHub," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015, Bergamo, Italy: ACM, 2015, pp. 805–816, ISBN: 9781450336758. DOI: 10.1145/2786805.2786850. [Online]. Available: https://doi.org/10.1145/2786805.2786850.

[41] S. Rai, R. C. Belwal, and A. Gupta, "A Review on Source Code Documentation," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 5, Jun. 2022, ISSN: 2157-6904. DOI: 10.1145/3519312. [Online]. Available: https://doi-org.lp.hscl.ufl.edu/10.1145/3519312.

[42] J.-P. Ostberg and S. Wagner, "On Automatically Collectable Metrics for Software Maintainability Evaluation," in *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, Rotterdam, Netherlands: Curran Associates, Inc., 2014, pp. 32–37. DOI: 10.1109/IWSM.Mensura.2014.19.

[43] D. Coleman, B. Lowther, and P. Oman, "The application of software maintainability models in industrial software systems," *Journal of Systems and Software*, vol. 29, no. 1, pp. 3–16, Apr. 1995. DOI: https://doi.org/10.1016/0164-1212(94)00125-7.

[44] GitHub, 2021. [Online]. Available: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes.

[45] T. Wang, S. Wang, and T.-H. (Chen, "Study the correlation between the readme file of GitHub projects and their popularity," *Journal of Systems and Software*, vol. 205, p. 111 806, 2023, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2023.111806. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223002017.

[46] X. Song, H. Sun, X. Wang, and J. Yan, "A Survey of Automatic Generation of Source Code Comments: Algorithms and Techniques," *IEEE Access*, vol. 7, pp. 111 411–111 428, 2019. DOI: 10.1109/ACCESS.2019.2931579.

[47] J. Li and I. Ahmed, "Commit Message Matters: Investigating Impact and Evolution of Commit Message Quality," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, Melbourne, Australia: IEEE, 2023, pp. 806–817. DOI: 10.1109/ICSE48619.2023.00076.

[48] W. R. Shadish and J. K. Luellen, "Quasi-Experimental Design," in *Handbook of Complementary Methods in Education Research*, 3rd. USA: Lawrence Erlbaum Associates, Inc., Apr. 2006, pp. 522–534.

[49] S. Osmanovic, *Project 3 template*, version 1.0.0, 2024. [Online]. Available: %7Bhttps://github.com/SaraOsmanovic/P3-template%7D.

[50] S. Osmanovic, *Introduction to GitHub - General GitHub Features*, https://youtu.be/SoEPYV6Nrxo, 2024.

[51] S. Osmanovic, *Local Development with Git and GitHub*, https://youtu.be/9MPMu8qBfPo, 2024.

[52] S. Osmanovic, K. Allen, and E. Hadam, *Mastering GitHub WorkFlow*, https://youtu.be/PbTdROv0TJw, 2024.

[53] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining GitHub for research and education: challenges and opportunities," *International Journal of Web Information Systems*, vol. 16, no. 4, pp. 451–473, Jun. 2020. DOI: https://doi.org/10.1108/ijwis-03-2020-0016.

[54] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, Hyderabad, India: ACM, 2014, pp. 92–101, ISBN: 9781450328630. DOI: 10.1145/2597073.2597074. [Online]. Available: https://doi.org/10.1145/2597073.2597074.

[55] G. Inc., *GitHub REST API documentation*, 2022. [Online]. Available: https://docs.github.com/en/rest?apiVersion=2022-11-28.

[56] D. L. Hahs-Vaughn and R. Lomax, *An Introduction to Statistical Concepts*. New York: Routledge, 2020, ISBN: 9781315624358. DOI: https://doi.org/10.4324/9781315624358.

[57] W. Bank, *Glossary — DataBank*, 2015. [Online]. Available: https://databank.worldbank.org/metadataglossary/gender-statistics/series/SI.POV.GINI.

[58] R. Vasa, M. Lumpe, P. Branch, and O. Nierstrasz, "Comparative analysis of evolving software systems using the Gini coefficient," in *2009 IEEE International Conference on Software Maintenance*, Edmonton, Alberta, Canada: IEEE Computer Society, 2009, pp. 179–188. DOI: 10.1109/ICSM.2009.5306322.

[59] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining Git," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, IEEE, Hyderabad, India: IEEE, 2009, pp. 1–10.

[60] N. Gitinabard, R. Okoilu, Y. Xu, S. Heckman, T. Barnes, and C. Lynch, "Student Teamwork on Programming Projects: What can GitHub logs show us?" In *Proceedings of The 13th International Conference on Educational Data Mining (EDM 2020)*, online: EDM, Jul. 2020, pp. 409–416.