Using GitHub in Large Software Engineering Classes
An Exploratory Case Study

Author 1, Author 2, and Author 3

Department
University
City, State, Zip

ARTICLE HISTORY
Compiled May 20, 2019

ABSTRACT
The online open source software (OSS) development platform, GitHub, has been recently used in Software Engineering (SE) classes to facilitate collaboration in student team projects. The underlying assumptions are that the technical and social features of GitHub can help students to communicate and collaborate more effectively as a team as well as help teachers to evaluate individual student contributions more objectively. To test these assumptions, in this paper, we explore the benefits and drawbacks of using GitHub as a means for assignment submission and project team collaboration in large SE classes. A case study is conducted in a senior level software engineering class with 91 computer science students divided into 18 project teams. Our research method includes entry and exit surveys, an exit interview, and a qualitative analysis of students' commit behavior throughout the period of the project. Our findings show that *a)* different teams adapted GitHub to their workflow differently, *b)* despite the steep learning curve, most teams managed to optimize their commit behavior over time without comprising the quality of their submissions, and *c)* in terms of using GitHub as a basis for student evaluation, our analysis exposed the risks of using such a platform for individual effort assessment. The work in this paper provides several valuable insights for researchers and makes several recommendations for educators about integrating Web-based configuration management tools in SE classes.

KEYWORDS
GitHub; Software Engineering; education

1. Introduction

Software Engineering (SE) has become an essential subject of Computer Science (CS) Curricula worldwide. According to the Accreditation Board for Engineering and Technology (ABET), an accredited SE class must provide both breadth and depth in all aspects of software development, from requirements gathering and system design, to software implementation and project management. In addition to these technical outcomes, the SE curriculum should include non-technical educational components that are designed to enhance the students' ability to function in teams and communicate effectively with a broad range of audiences. The main objective is to equip students with a set of soft and technical skills that are necessary to attain a successful career in software engineering after graduation.

To realize these outcomes, most core SE classes include some sort of a mid-size team project that students have to work on during the class. The educational objectives of the team project are to *a)* reinforce the concepts being taught in the classroom, and *b)* simulate an industrial software engineering environment (Coppit & Haddox-Schatz, 2005; Hayes, Lethbridge, & Port, 2003; Ohlsson & Johansson, 1995). However, due to the limited time frame (typically 4 months) and the general lack of industrial experience of undergraduate students, such projects often pose many challenges for students and teachers, especially in relatively large classrooms (50+ students). These challenges are typically related to problems with inter-team communication and collaboration as well as the evaluation of individual student contribution (effort) (Coppit, 2006; Gates, Delgado, & Mondragón, 2000; Hayes et al., 2003).

Recent research has exposed several communication problems for students working in teams (Liu, 2005). In general, mainly due to the lack of proper teamwork training, most undergraduate students struggle with basic communication skills, or do not even recognize the value of establishing and sustaining an effective communication channel with their teammates. Other problems arise from the logistical hurdles typically associated with the conflicting schedules of undergraduate students (cannot agree on a time or location to meet) and the lack of a unified platform, or tool, of communication that all team members can effectively use (Chao, 2007; Goold, Augar, & Farmer, 2006; King & Behnke, 2005; Seppälä, Auvinen, Karavirta, Vihavainen, & Ihantola, 2016).

Another major challenge facing students working in teams is inter-team collaboration (Colbeck, Campbell, & Bjorklund, 2000; Hansen, 2006; Johansson, 2000). Despite being encouraged otherwise, students often end up forming teams of other students whom they feel comfortable working with (mainly friends), rather than based on technical merits. This leads to the formation of unbalanced teams in terms of technical and soft skills. In most cases, unbalanced teams lead to the emergence of *cowboy programmers* (Curtis, 2001; Hollar, 2006), where one dominant person in the team does all the work, while the rest struggle to maintain the same level of contribution. Consequently, this leads to the emergence of *free-riders*, or team members who lost interest in the project and are satisfied with not being active in their teams (Buchta, Petrenko, Poshyvanyk, & Rajlich, 2006; Coppit & Haddox-Schatz, 2005; Slavin, 1980; Williams, Beard, & Rymer, 1991).

From an educational point of view, a key challenge facing teachers in software team projects is to come up with objective methods to evaluate individual team members. This challenge stems from the fact that individual contributions are typically not separately quantifiable. Several grading schemes have been proposed in the literature to deal with this problem. In general, these schemes can be categorized into two types: individual student grading and the one-grade-fits-all approach (Clark, 2005; Clark, Davies, & Skeers, 2005; Coppit & Haddox-Schatz, 2005; Hayes et al., 2003). Peer evaluation (students are asked to evaluate each other's performance) and self-evaluation (students are asked to submit a report detailing their specific contributions to the project) are the two most popular methods for individual evaluation. In the one-grade-fits-all approach, one grade is assigned to the entire team based on their overall performance as a team. However, due to the power dynamics within the team, personal relationships, and in some cases racial and gender biases, these approaches often fail to produce an objective evaluation of individual students' contribution (Hayes et al., 2003; Wilkins & Lawhead, 2000).

In an attempt to overcome these challenges, online version control systems, such as *GitHub* and *SourceForge*, have been recently utilized in software engineering classes to facilitate student collaboration in team projects. Such platforms provide programmers with a set of online services and tools to host, share, and maintain their code as well as create world-wide social networks of developers at unprecedented scales. The unique set of technical and social features such platforms support have made them an appealing tool to be used as a means to facilitate student communication, collaboration, and evaluation in class-based software team projects (Feliciano, Storey, & Zagalsky, 2016; Zagalsky, Feliciano, Margaret-AnneStorey, Zhao, & Wang, 2015).

Several early studies have been conducted on utilizing GitHub in SE class projects (Feliciano et al., 2016; Kertész, 2015; Zagalsky et al., 2015). Despite this effort, there is still a lack of empirical evidence on the benefits and drawbacks of this approach, especially in relatively large-size classrooms. To bridge this gap, in this paper, we present the results of an exploratory case study on using GitHub in SE class projects. According to Wohlin et al. (2012), *a case study is conducted to investigate a single entity or phenomenon in its real-life context, within a specific time space*. In our case study, the phenomenon of interest is enforcing *GitHub* as a collaboration platform in SE student team projects, the context is the CS 0000 class offered by the Computer Science Department at Alpha University, and the time-frame is the Fall semester of 2016. Our main objective is to explore the impact of using such a platform on the individual and aggregate performance as well as collaborative behavior of students. Case studies can be powerful tools for establishing early knowledge in unexplored domains. Such knowledge can be utilized to formulate formal hypotheses, and eventually, build a unified theory for the domain of interest. Our contributions in this paper are:

- We conduct entry and exit surveys to measure the impact of enforcing GitHub in SE classes and outline the main challenges faced by students using such a platform for collaboration and assignment submission.
- We make several recommendations for instructors about enforcing GitHub in large SE classes and describe the risks associated with using such a platform for evaluating students' effort or their contribution to the project.
- By analyzing students' commit patterns, we provide valuable insights into students' behavior when utilizing GitHub. These insights suggest further empirical investigations to formally understand the impact of using such a tool in educational settings.

The rest of this paper is organized as follows. In Section 2, we review seminal related work and outline our motivations. In Section 3, we describe our case study's setup. In Section 4, we describe the entry and exit surveys and exit interview and analyze their results. In Section 5, we present and discuss the results of our qualitative analysis of students' commit behavior. In Section 6, we provide a set of recommendations for teachers based on our main findings. In Section 7, we discuss the potential threats to our study's validity. Finally, in Section 8, we conclude the paper and outline several prospects of future work.

2. Related Work

In this section, we review seminal work related to student collaboration and evaluation in SE team projects, summarize existing work on the utilization of GitHub in educational settings, and outline our main motivations and research questions.

2.1. Team Projects in SE Courses

Students' collaboration and evaluation in SE class projects have received considerable attention in the literature. In particular, researchers have focused on the most effective teaching strategies for facilitating teamwork and objectively evaluating individual student effort. For instance, in an attempt to narrow down the gap between class and industrial practices, Buchta et al. (2006) developed a course where students practiced software evolution through the implementation of change requests on medium-sized open-source software systems. A Concurrent Versioning System (CVS), that was specifically developed for the class, was used to coordinate teamwork. Students were asked to submit a report after the completion of each phase and an assessment survey was conducted at the end of the semester to get students to rate their experience. The results showed that adapting an incremental change format in SE class projects helped to address several problems related to individual student accountability and increased students motivation as well as their understanding of the software engineering process.

Chao (2007) explored the potential uses of *wikis* to facilitate team collaboration and communication in student projects. Mainly, the authors sought to compare the effectiveness of team communication and collaboration using *wikis* versus more traditional communication mechanisms such as email and discussion boards. The authors reported that students quickly discovered a number of innovative ways in which *wikis* could augment collaborative software development activities, such as project planning, requirements management, and effort tracking. An anonymous survey at the end of the project revealed that the vast majority of students found *wikis* to be a good tool for project collaboration.

In an attempt to provide a more realistic project experience for students, Coppit and Haddox-Schatz (2005) presented an approach to teaching a one-semester large-scale software engineering course in which students worked together to construct a moderately sized software system. The proposed approach included multiple strategies for facilitating scheduling, project management, communication, and development, at a large scale. The authors also implemented a system for automatically computing the project grade for each student in the class based on a predefined project point system. While the overall experience was positive, the authors pointed out several challenges regarding the choice of the project as well as the integration of under and over-achieving students in large scale teams.

Hayes et al. (2003) tackled the challenge of fairly and accurately discerning the effort of individual students in SE team projects for evaluation purposes. Specifically, the authors presented and discussed several grading approaches and best practices for evaluating students' contributions. The authors made several recommendations to ensure the fairness and consistency of the grading process. These recommendations included, for instance, using project demonstrations and quizzes to further test the students' knowledge of their projects, allowing team members to evaluate each other, and to carefully and frequently monitor this process to prevent the mob mentality among students.

Goold et al. (2006) investigated the use of an online learning environment platform to enhance students' experience when working in virtual teams. Three anonymous surveys were conducted to elicit feedback from students about their experiences in working in virtual teams within the learning environment. Most students indicated that they valued the opportunity to discuss various aspects of the course with peers and teaching staff online as well as to interact with real-life employees. The students also reported that online team work provided the flexibility of time and place and allowed communication and participation to be recorded. However, problems were reported when team members left participation and submission to the last minute.

Clark et al. (2005) tackled the challenge of assessing individual contributions and performance in SE class team projects. Specifically, the authors experimented with a suite of Web-based peer assessment tools. The suite supported a time-sheet, a self/peer evaluation survey, an individual contribution report, and a quantity report. These tools allowed students to self-evaluate their contributions as well as other students' contributions to the project. Different performance indicators from these tools were then used to calculate the final grade of each student. The authors concluded that the proposed suite provided timely feedback to students and enabled the lecturer to manage the assessment of larger and more diverse student cohorts.

2.2. Using Git/GitHub as Educational Tools

Motivated by its undeniable positive impact on the Open Source Software (OSS) movement, along with its social and technical features, GitHub has been recently utilized in SE and programming classrooms as a tool for managing student projects. This has encouraged researchers to further investigate the benefits, drawbacks, and challenges associated with using GitHub in the classroom. For instance, Zagalsky et al. (2015) examined how GitHub could improve or possibly hinder the educational experience of students and teachers. In particular, the authors conducted a qualitative study to understand how GitHub was being used in education, and the motivations, benefits and challenges it brought. The study consisted of analyzing online posts describing personal experiences in using GitHub in the classroom along with several interviews with faculty who used GitHub to support teaching or learning. The analysis revealed that GitHub was mainly utilized in classrooms as a submission and hosting platform. Furthermore, the transparency of GitHub encouraged students to participate and contribute more to the hosted course material. The list of limitations included the barriers to entry, long learning curve, and the general lack of direct support for popular educational formats, such as PDF and LaTeX.

In a follow-up study, Feliciano et al. (2016) examined students' perspectives on using GitHub as an educational platform. The authors conducted a case study over two classes in which GitHub was used for class material dissemination, lab work submission, and project hosting. The study design included direct interviews with the students followed by a validation survey. The results showed that GitHub promoted student cooperation and cross-team collaboration, making students more involved in the course. In addition, students were able to develop and demonstrate industry-relevant skills. However, students have raised several concerns about having their work publicly available, the steep learning curve of Git and GitHub, and the lack of educational features to support grading and assignment management.

Kertész (2015) investigated the merits of using GitHub as a collaborative platform for students to do their homework and submit their classroom assignments. The results of analyzing students commit patterns along with the results of an exit survey indicated that, in general, students found GitHub to be useful in learning from their faults and getting help from colleagues much faster. In addition, students appreciated the opportunity of using a platform that is commonly used in the industry. In terms of challenges, students pointed out a steep learning curve and low activity levels by those not familiar with the platform.

Haaranen and Lehtinen (2015) described how to incrementally present the features of Git and incorporate them into a CS course's workflow. In particular, the authors presented a case study of running a large Web software development class utilizing Git. Data was collected using a mixed approach, combining a feedback survey after individual exercises, an exam that was completed by the students, and the Git usage data. The results showed that Git could be used successfully to disseminate course materials and facilitate exercise submissions. Furthermore, the results showed that enforcing Git in the classroom helped students to acquire a set of essential skills desired by the industry. However, several concerns were raised about the difficulty of learning Git and its suitability as an educational platform.

Heckman and King (2018) introduced a framework for supporting software engineering practices in CS classrooms through the use of Eclipse for development, GitHub for submission and collaboration, and Jenkins for continuous integration and automated grading. The framework was evaluated through multiple case studies, involving five undergraduate CS core courses. The result showed that the proposed framework, combined with GitHub and Jenkins, helped in automated grading and in exposing students to professional software development tools.

Hsing and Gennarelli (2019) investigated the impact of using GitHub on several key learning variables in CS classrooms. The authors surveyed 7530 students and 300 educators from GitHub and non-GitHub classrooms. The results showed that using GitHub in the classroom predicted better learning outcomes and classroom experiences. For instance, students who received instructor feedback via GitHub reported benefiting more from the feedback. Furthermore, students who used GitHub in the classroom reported that they learned more about teamwork and collaboration and felt more prepared for a future internship and career.

2.3. Summary, Motivation, and Research Questions

Our review has revealed that, in general, the assumptions behind utilizing GitHub in SE class projects fall under the tenets of the collaborative learning theory. This theory describes situations in which two or more people build synchronously and interactively a joint solution to a specific problem. The theory suggests that learning is inherently a social process, thus, it emphasizes the extent and quality of the exchanges that occur within teams of students in collaborative environments as a way for increasing critical thinking and team spirit (Curtis, 2001; Gokhale, 1995). GitHub promotes *social coding* - an idea that combines programming and social features, such as user profiles, newsfeed, following repositories, and code sharing. These features are designed to enable developers to exchange information more freely and in the open and build social networks of programmers working toward the same goal. Therefore, enforcing such a platform in class is expected to enhance students' collaboration and their sense of teamwork. Furthermore, GitHub's transparency can motivate students to self-regulate their contributions by observing the progress of their teammates as well as other teams in class, thus enhancing their productivity and reducing their proneness to academic procrastination (Klassen, Krawchuk, & Rajani, 2008; Steel, 2007).

Another objective of enforcing a tool such as GitHub in the classroom is to prepare students for their future careers. Specifically, while most SE curricula typically cover concepts of configuration management, due to time limitations, students often receive limited exposure to the different configuration management platforms commonly used in industrial settings. However, by using GitHub as the main platform for managing their term project, students can get a hands-on experience using such a platform in semi-professional settings (Buffardi, 2015).

In terms of limitations, our review shows that the most common challenges that limit the utilization of GitHub in SE classes are: the steep learning curve often associated with introducing a new technology in the classroom, the lack of features to support class-specific tasks, such as assignment submission and grading, and the conflicts that typically arise from the variation in GitHub experience among students. Our review also shows that multiple studies have examined using GitHib's tracking features as a basis for student evaluation (Haaranen & Lehtinen, 2015; Kelleher, 2014; Kertész, 2015). In general, the evaluation of individual contributions in team projects can be challenging as it is often hard to distill individual contributions to a shared project. Using GitHub, individual students can be evaluated based on their contributions, such as the number and/or size of their commits, pull-requests, and commenting on fellow students' code. Such information is typically combined with peer and self-evaluation mechanisms, or a subjective assessment of contribution quality, to enhance confidence in the grade (Kelleher, 2014).

In summary, our review shows that GitHub can potentially improve the teaching and learning experience in SE classrooms. However, there is still a research gap on how such a platform actually affects students' configuration management skills and their overall project team dynamics, especially in large classroom settings (50+ students). To bridge this gap, in this paper, we explore through a case study how students utilize GitHub in their team projects along with the main limitations and drawbacks associated using such a platform in educational settings. Case studies are necessary to explore a phenomenon before formal experiments can be run and a theory can be developed. They can be particularly useful when the outcome of the research is highly dependent on the context of the study, which is mainly the case in most educational research (Wohlin et al., 2012). The main objectives of our case study are: to provide insights into how students would integrate GitHub into their workflow, to assess the validity of using such a platform as a basis for evaluating individual student contribution, and to analyze the impact of using such a tool on the quality of students' work, their communication and configuration management skills as well as other academic behaviors such as procrastination. To guide our analysis, we formulate the following research questions:

- **$\mathbf{RQ_1}$.** What are the main challenges and benefits of using GitHub in SE student team projects?
- **$\mathbf{RQ_2}$.** How do students utilize the technical and social features of GitHub in SE team projects?
- **$\mathbf{RQ_3}$.** Can GitHub be used as a basis for evaluating individual student contribution?

3. Case Study Setup

CS 0000—Software Systems Development, is a core senior-level software engineering class offered by the Department of Computer Science at Alpha University. This class covers the main phases of the software engineering process, focusing on concepts of project management, requirements engineering, and software testing. The class has a significant semester-long mid-size software project component that students are expected to execute in order to pass. Even though it is not officially a capstone design class, CS 0000 acts as a final senior project class for students in the Software Engineering concentration offered by the CS department. At the beginning of the semester, students are asked to form their project teams and choose their projects. Each team should consist of 4-5 students. The project is divided into 5 assignments. These assignments can be described as follows:

- Software Requirements Specification (SRS): for the first assignment, students are required to gather and document the main functional and non-functional requirements of their systems. Students are expected to use the IEEE 830-1998[1] standard's template to prepare their SRS documents. The functional and non-functional requirements of the system are described using textual use cases. Students are encouraged to use any form of visual aid (e.g., use case diagrams and sequence diagrams) to further describe their requirements.
- Software Design Document (SDD): for the second assignment, students are required to define and describe the main modular components of the system and their relations. This multi-view document should include a software architecture diagram, a database schematic diagram (in case the system supports a database), a folder structure of the system, and the hardware view of the system (Clements et al., 2000). There is no specific template for this assignment. However, students are encouraged to follow the IEEE 1471-2000[2] standard's guidelines for documenting software architecture.
- Software Testing Document (STD): for this assignment, students are required to describe their test plan and design a set of test cases for their system. Students are expected to write a separate test case for each feature they listed in their SRS document. Students are encouraged to follow the IEEE 829-2008[3] standard for documenting their test cases.
- Code: at the end of the semester, students in CS 0000 are required to submit a working copy of their project code for grading.
- Project Management Document (PMD): this is an active document that is assigned at the beginning of the semester and submitted after the final project presentation. The students are supposed to document the logistic aspects of their project, including a summary of their meetings, configuration management plan, potential commercialization plan, and risk management plan.

Students are free to adapt whatever software development process model (software life-cycle) they find appropriate. However, most teams end up with a mixed-model where code is produced in Agile format, while other project documents are procedure in a Waterfall format, following their assignments due dates. The deadline for each assignment is exactly two full weeks (14 days) after the initial assignment date. Students are expected to spend around 65% of class time to work on the project. The project is worth 65% of the grade. The remaining 35% is divided between a midterm exam and a final exam.

At the end of the semester, each team has to present their final product to an audience of industry professionals, academics, and other students in the class. Each team is given 10 minutes, during which, each team member has to present a part of the project and talk briefly about their specific contribution. In addition to a working demo, the presentation must include five other main components: introduction, problem, proposed solution, tools used, and team members. Students are graded based on

[1] `https://standards.ieee.org/standard/830-1998.html`

[2] https://standards.ieee.org/standard/1471-2000.html

[3] `https://standards.ieee.org/standard/829-2008.html`

the quality of their talk, including following the presentation guidelines and running a fully working demo. Furthermore, students are judged based on their soft skills, such as showing up on time, following the business-casual dress code, and the ability to answer audience questions.

In the Fall of 2016, at the beginning of the semester, the class had 91 students, divided into 18 project teams. The students were informed that GitHub would be the only method to submit the project assignments. The documentation assignments (SRS, SDD, and STD) had to be submitted using MarkDown, a lightweight markup language with plain text formatting syntax that is typically used to create GitHub ReadMe files. Submissions were graded based on the last commit made to the assignment before 11:55 PM of the day at which the assignment was due. It is important to point out that the third author of the paper has taught CS 0000 twice before (the Fall semesters of 2014 and 2015). However, GitHub was only used for the first time during the Fall semester of 2016 (the semester during which this case study was conducted).

The teaching assistant (TA) of the class held a tutorial to introduce students to GitHub at the beginning of the semester. The tutorial included an introduction to Git as a version control system that is used to manage source code history. GitHub was then introduced as one of the online hosting services for Git repositories. Students were then introduced to *GitHub Desktop*, a cross-platform client for Git. GitHub Desktop provides an easy GUI to help developers contribute to projects (branch off, merge, and deploy) without the need for using the command line of Git. All students in our class reported using GitHub Desktop for their projects. The tutorial also included introducing students to the basic concepts of configuration management and version control, such as creating a repository, commits, comments, pull requests, merge, forks, and branches. In addition, students were given instructions on how to create basic MarkDown documents, including tables and figures. The students were further encouraged to watch multiple GitHub tutorial videos that were recommended by the instructor.

Students were assured that their final project grade would not depend on their level of activity on GitHub (i.e., number of commits and comments). In other words, students were told that they were free to adopt any commit strategy they felt comfortable with as a team. Our main objective was to track how different teams of students would utilize such a platform in the absence of an evaluation component.

The students were given the freedom to choose their projects and whatever technologies and tools they wanted to work with. To ensure that all teams had projects of sufficient breadth and depth, no video-games or single feature mobile applications were allowed. All projects were approved by the teacher and the TA at the beginning of the semester. The students were presented with several team formation options, including ego-less (all team members share equal responsibilities), chief-programmer, and hierarchical structures. However, the structure of each team was left for the students to decide. Unfortunately, there was no designated lab for the class. The students were expected to meet outside of class to organize their teams.

The research methods used in our case study included entry and exit surveys, an exit interview, and a qualitative analysis of students' commit patterns. These methods are commonly used to collect data in case-study research (Wohlin et al., 2012). In what follows, we describe each of these methods along with our main findings in greater detail.

4. Survey Design and Results

Surveys are commonly used in empirical studies to obtain a quick snapshot of the current status of a target population (Ciolkowski, Laitenberger, Vegas, & Biffl, 2003). In general, surveys can take the form of direct interviews or written questionnaires. While interviews can help to elicit more thorough and honest responses from subjects, questionnaires have the advantage of being cheaper to execute, especially when the population is relatively large. In software engineering research, surveys have become a standard tool for data collection (Punter, Ciolkowski, Freimut, & John, 2003). Interview and questionnaire surveys are frequently conducted to gather rapid feedback from software engineering practitioners on a variety of topics (Lo, Nagappan, & Zimmermann, 2015; Manotas et al., 2016). Furthermore, surveys are commonly used in classroom research to elicit students' feedback toward new teaching strategies (Buchta et al., 2006; Chao, 2007).

In our case study, we used a questionnaire-type survey to collect pre-and-post treatment data from the subjects. The decision to use questionnaires allowed our subjects to remain anonymous. In classroom surveys, anonymity can improve the response rate and enhance the validity of the study by obtaining less biased and more objective information from students. Specifically, students might be reluctant to express their true opinion out of the fear that a response that is not aligned with the expectations of the teacher would affect their grades or the teacher's attitude toward them (Carver, Jaccheri, Morasca, & Shull, 2010; Reisbig, Jr, White, & Rush, 2007).

Our study included two anonymous surveys, a descriptive entry survey and an exploratory exit survey, and an exit interview. The entry survey was used to collect general descriptive information about the population. The exit survey was used to explore how the applied treatment (i.e. enforcing GitHub in the class) influenced students' experience (Carver, Jaccheri, Morasca, & Shull, 2004). Finally, the exit interview was conducted to elicit final face-to-face feedback from students. In what follows, we describe these surveys and their results in greater detail.

4.1. Survey Design

The entry survey was conducted at the beginning of the semester. The population consisted of 91 students: 47 juniors, 43 seniors, and 1 sophomore. The purpose of the survey was to collect initial descriptive data about the population, including the students' prior experience in programming and their familiarity with GitHub as well as other configuration management platforms. The TA handed out the written questionnaires to the students to fill out. The instructor was not present in the classroom during the survey. This step was necessary to remove any bias that would result from the teacher's presence. The students were assured that the survey was anonymous and were encouraged to be as honest as possible in their responses. The questions in the survey are shown in Table 1. The response rate was 100% (i.e., all of the students in the class completed the survey). The survey results were transcribed and coded using *Microsoft Excel*, and then analyzed using *IBM SPSS* statistical package.

The exit survey contained the same questions as the entry survey and an additional open-ended question to collect students' perceptions of using GitHub in an SE course (*"Did you face any challenges using Github for this class?"* The exit survey was completed by 84 students: 45 juniors, 38 seniors, and 1 sophomore. The response rate was 92% (3 students had dropped the class and 4 did not complete the exit survey). The

Table 1.: The questions in the entry survey

Question	**Answer Variants**
What year are you?	Junior / Senior
How many programming languages do you know?	Numeric
How many years of experience as a programmer do you have?	Numeric
How familiar are you with configuration management systems?	a. Very familiar, experienced b. Familiar c. Somewhat familiar d. Not familiar
Do you have a GitHub account?	yes / no
How experienced are you with GitHub?	a. Very experienced b. Experienced c. Somewhat experienced d. Never used before
Are you familiar with other platforms? Please indicate your level of experience for each of the platforms below from 1 (never used) to 4 (very experienced).	a. SourceForge b. DropBox c. BitBucket d. Other (please specify)

results were also coded in the same manner and analyzed using *IBM SPSS*. In what follows, we describe the main outcomes of the surveys.

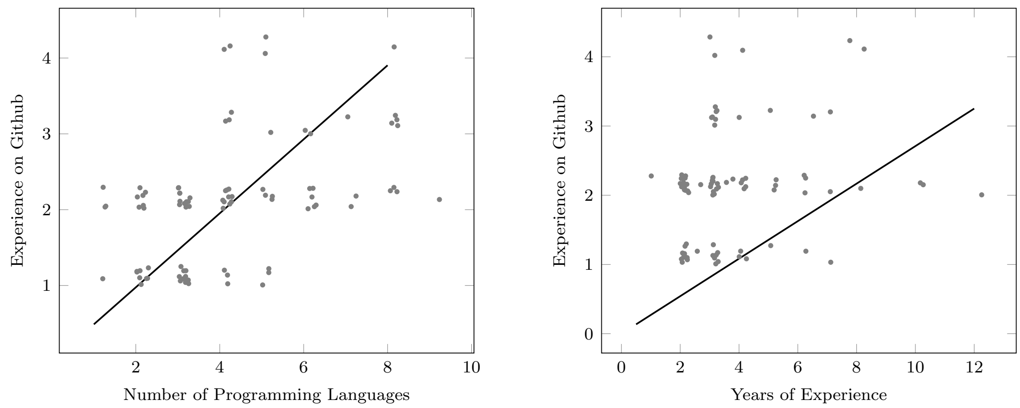
4.2. Entry Survey Results

Table 2 presents the results of the entry survey. The results show that student cohort was split almost equally between 3rd (junior) and 4th-year (senior) students. As expected, juniors were less experienced with programming and knew fewer programming languages than seniors.

Table 2.: Descriptive statistics for the entry survey: the number of programming languages students knew, their experience on GitHub, and years of programming experience as reported by our study participants

		Prog Languages				**Exp. on GH**				**Years of Exp.**			
	N	Mean	Med.	StD	Range	Mean	Med.	StD	Range	Mean	Med.	StD	Range
All students	91	4.06	4.00	1.95	(1-9)	1.94	2.00	0.79	(1-4)	3.62	3.00	2.05	(1-12)
Junior	47	3.60	3.00	1.62	(1-8)	1.94	2.00	0.82	(1-4)	3.45	3.00	1.92	(2-10)
Senior	43	4.49	4.00	2.11	(1-9)	1.91	2.00	0.72	(1-4)	3.71	3.00	2.11	(1-12)

Correlation analysis is commonly used in exploratory case studies as an effective and straightforward tool to reveal any relationships in the data (Runeson & Höst, 2009). It helps to gain in-depth insights into the population as well as the variables under study. In our analysis, we used Spearman's correlation to examine the relationship between students' experience on GitHub and the overall programming experience



(a) Number of Programming Languages vs. GitHub Experience, ($R = 0.488, p < 0.001$)

(b) Years of Experience in Programming vs. GitHub Experience, ($R = 0.271, p < 0.01$)

Figure 1.: The results of Spearman's correlation. Noise was added for overlapping points

of students. The results, in Fig. 1-a, show a positive and statistically significant relationship between GitHub experience and the number of programming languages a student knows (R=0.488, p < 0.001). In addition, Fig. 1-b shows a statistically significant relation between GitHub experience and programming experience in years (R = 0.271, p < 0.001). Overall, the results indicate that a more experienced student in programming (number of years and number of programming languages) is more likely to be more experienced in a platform such as GitHub.

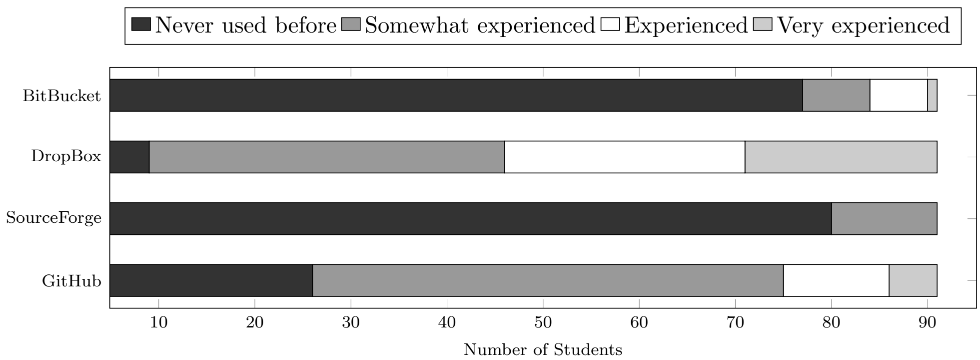


Figure 2.: Students' experience with GitHub and other platforms

Fig. 2 breaks down students' experience with GitHub, SourceForge, Dropbox, and BitBucket. Most of the students reported high experience with Dropbox, followed by GitHub. Around a half of the students indicated that they were somewhat experienced with GitHub and a fourth of the students reported no knowledge of GitHub. The majority of students reported that they never used SourceForge or BitBucket before.

4.3. Exit Survey Results

The plots in Fig. 3 show the GitHub experience as reported by the students in the entry and exit survey. In the entry survey, only 11 students identified themselves

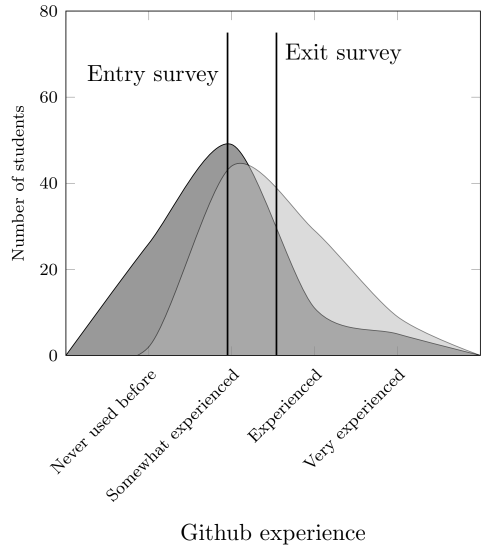


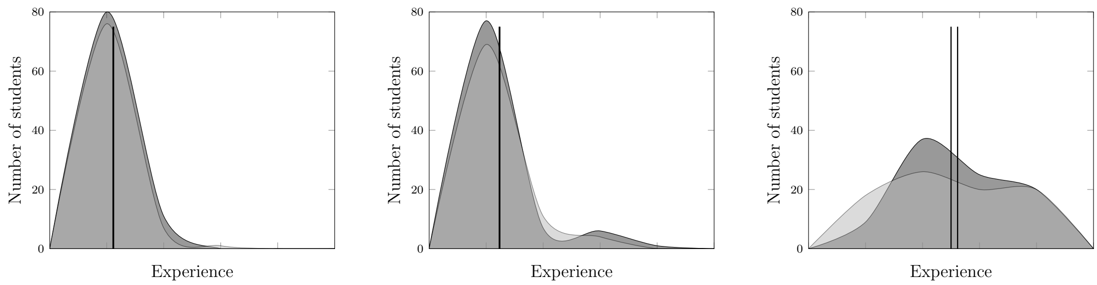
Figure 3.: Frequency distribution comparison of GitHub experience () in the entry and the exit surveys

as *"Experienced"* and even fewer as *"Very experienced"* as opposed to *"Somewhat experienced"* and *"Never used before."* The mean is in the *"Somewhat experienced"* category. In the exit survey, the number of *"Somewhat experienced"* decreased from 49 to 44, but the number of more experienced students increased drastically, especially in the *"Experienced"* category, from 11 to 29. The mean for the exit survey increased, settling between *"Somewhat experienced"* and *"Experienced."*

To test for statistical significance, we used Wilcoxon Rank Sum test. This test is non-parametric; it makes no assumptions about the distribution of the data. Thus, it can be used when comparing two teams by continuous or ordinal non-normally distributed dependent variables. The results of the Wilcoxon test showed that the difference in GitHub experience for all students was statistically significant ($Z = -4.504, p < 0.001$).

Fig. 4 compares students' experience in other popular CM platforms before and after the class as reported in the entry and exit surveys. In general, the shaded areas for SourceForge, BitBucket, and DropBox overlap, which means that students' experience in other platforms remained almost the same. Wilcoxon Rank Sum test showed that the difference between entry and exit survey means was not statistically significant ($Z = -0.535$ for SourceForge, $Z = -0.968$ for BitBucket, and $Z = -0.727$ for DropBox respectively). Overall, we conclude that enforcing GitHub in class did not enhance experience in other CM platforms. It is important to point out that our findings here are based on a subjective self-assessment measure of students' experience. Therefore, while these results give an indication on the direction of the relation, they should be interpreted with care.

The last question in the exit survey asked students to share any challenges they faced while using GitHub during the semester. Students' responses were qualitatively analyzed using a systematic coding of the data (Wohlin et al., 2012). Specifically, the coding process involved each of the authors individually going through the set of free-form answers, identifying the main response categories as they appeared in the text. The categorization of each author was then manually examined by the other two



(a) SourceForge (b) BitBucket (c) DropBox

Figure 4.: Frequency distribution comparison of experience in other CM platforms in the entry and the exit surveys

Table 3.: A summary of the main challenges of using GitHub as described by the students in the exit survey

Challenges	**Count**
NA	47
Resolving merge conflicts	9
Learning curve	6
Working on the same file	5
Markdown	4
Branching	3
Other	10

authors. A discussion session was then held to resolve any conflicts and to make sure that the categories were exhaustive and mutually exclusive. The majority of conflicts in individual classifications were related to the granularity level, or abstraction, of the category. For example, two out of the three students considered difficulties in learning Markdown to be a separate response category rather being included under the "Learning curve" category. The rationale was that the word "Markdown" had explicitly appeared in multiple answers, thus, should be considered separately. Such conflicts were resolved using majority voting.

The outcome of the coding process (i.e., specific response categories) for the last question of the exit survey is presented in Table 3. The table shows that out of 84 students, 47 mentioned that they did not experience any difficulties using GitHub. The difficulties ($\mathbf{RQ_1}$) that were commonly reported by the students could be described as follows:

- **Resolving merge conflicts**: several students reported facing problems when resolving merge conflicts. This apparently was a common issue in the documentation assignments, especially toward the deadlines where most students started submitting their updates. This issue could have been resolved by holding another tutorial before the first assignment to explain to students the best way for resolving merge conflicts. Actually, one of the students pointed that out in her answer: *"I feel like more attention should be placed on teaching students how to resolve merge conflicts and dealing with branches."*
- **Steep learning curve**: similar to what have been observed in related liter-

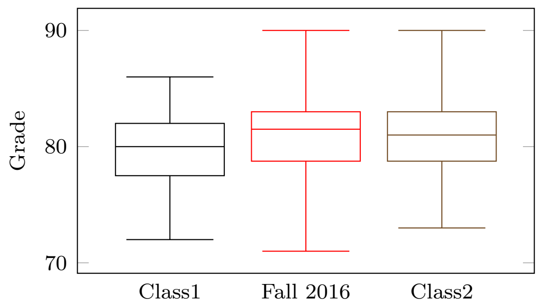


Figure 5.: Comparing the grades (quality of submission) from the Fall 2016 to two other sections of CS 0000 where GitHub was not used

ature (Feliciano et al., 2016; Kertész, 2015; Zagalsky et al., 2015), the steep learning curve was an issue for some students. These were mainly the students who had no prior experience with GitHub, or any other online version control systems, before CS 0000. Students expressed their concerns about this issue using statements such as *"Bit of learning curve its a bit complicated"* and *"GitHub is hard to understand and use."*

To determine if the overhead of this steep learning curve had impacted the quality of students' work, we compared students' grades with the average grades from two previous sections of CS 0000 (Fall semesters of 2014 and 2015) in which GitHub was not used. The results in Fig. 5 provide evidence that the overhead that resulted from using GitHub in class did not impact the quality of students' work. It is important to point out that using grades as a proxy for assessing the quality of students' work might raise some construct validity concerns. Specifically, grades can be subjective, especially in assignments such as the SRS and SDD where there is no wrong answer. In an attempt to control for this effect, the assignments were graded by the same instructor and the same TA and using the exact same rubric used before in the other two classes. Therefore, these concerns were minimized.

- **Technical difficulties**: several students reported unexpected problems with the platform. For instance, some students noted that GitHub sometimes did not save changes if several students were working on the same file simultaneously: *"changes on the same document being done at the same time, causing some loss of changes from one of them"* and *"while multiple of members were modifying a project, it deleted a team member's work."*
- **MarkDown**: a small number of students (4/84) reported difficulties when dealing with MarkDown, especially when formatting figures and tables. These concerns appeared in comments such as *"I faced a few issues with formatting in MarkDown."* Consequently, some students have suggested using other, easier to format, platforms such as Google Documents, for example, *"I feel like google docs is a better option"* and *"there are simply better options for the document submissions."*

4.4. Exit Interview Results

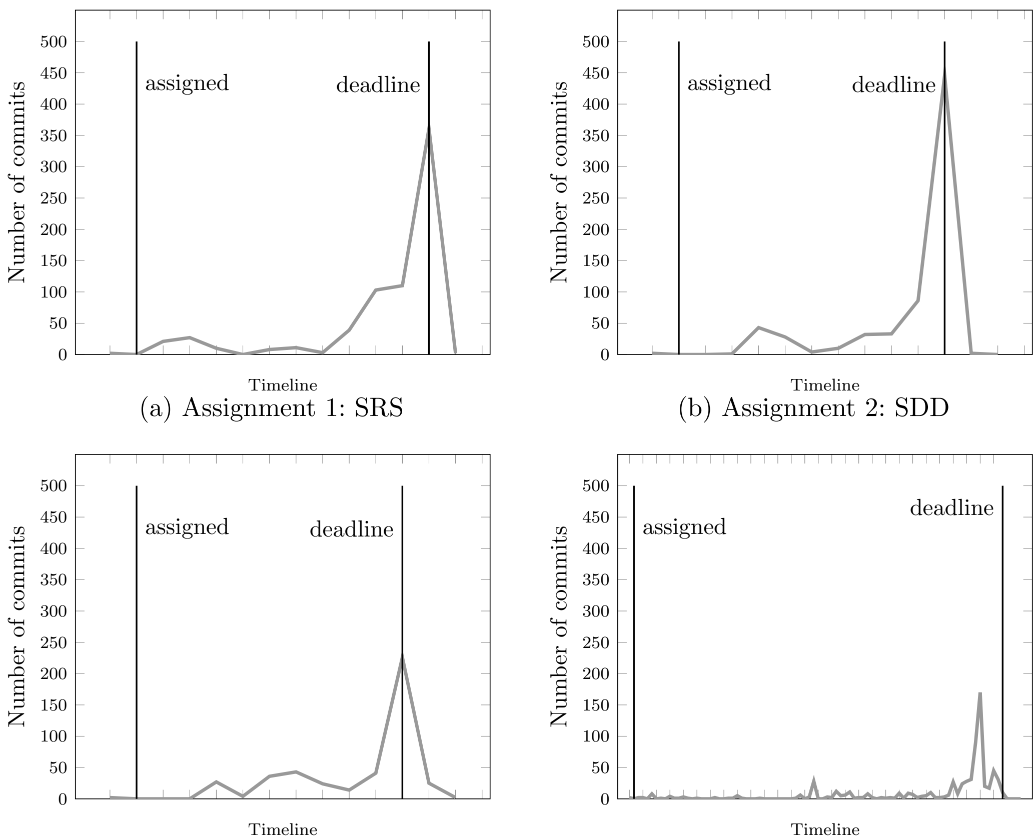
In addition to the anonymous exit survey, an exit interview was conducted after the final copy of the project was submitted. Specifically, all teams were required to do a live demo separately. During this demo, students were asked to describe their overall experience with using GitHub in the class. The interviews were semi-structured (Wohlin et al., 2012), all teams were asked the same question (i.e., *"Please describe your overall experience with using GitHub in class"*), and the discussion then followed the different topical trajectories in the conversation. Each interview session lasted between 10-15 minutes. Students' answers were later coded following the exact coding process used for the open-ended question in the exit survey. Overall, the coding of the interview responses resulted in three response categories. These categories can be described as follows:

- **Experience with GitHub:** all teams in class indicated that enforcing GitHub as a mandatory configuration management platform forced them to learn about the platform. Several teams (7/18) indicated that they probably would have not used the platform if it were optional. Overall, the students reported feeling more confident using GitHub and more familiar with CM concepts after class ($\mathbf{RQ_1}$).
- **Steep learning curve:** almost all teams confirmed the steep learning curve, especially at the beginning of the semester when not all team members were familiar with the platform. Several teams indicated that it took them until the second assignment to fully understand how the platform worked and how to resolve conflicts ($\mathbf{RQ_1}$). Furthermore, all teams in class have indicated that they used *GitHub Desktop*[4] to avoid dealing with the command line of Git.
- **Team Structure:** a few teams (4/18) reported that GitHub negatively impacted team structure, including issues of cowboy programming and free riders. These phenomena emerged in two teams where one member had more prior experience with GitHub that the rest of the team. This encouraged that person to put on the cowboy hat. Other team members, did not feel the need to commit as the cowboy *"took care of that"*, even though the team was relatively balanced in terms of other technical skills. The rest of the teams indicated that using GitHub helped them to manage the team better, especially in task assignment and progress tracking issues ($\mathbf{RQ_1}$ and $\mathbf{RQ_2}$).
- **Team communication:** majority of the teams (14/18) implied that they did not feel the need, or even see the value, of using the social features of GitHub to facilitate more communication. In general, being in close geographical distances (mainly living on or around the campus), the students felt that the social features of GitHub that facilitates distributed team management were unnecessary. The lack of effective mobile support was also mentioned by some students as they compared GitHub communication tools to other, more user friendly platforms such as the social networking app GroupMe ($\mathbf{RQ_1}$ and $\mathbf{RQ_2}$). Few students expressed concerns about their repositories being public. They were worried about their discussions to be perceived negatively by the TA or the instructor. Therefore, they preferred private communication mediums such as GroupMe.

5. Analyzing Students' Commit Behavior

In this section, we aggregate and analyze students' GitHub activities over the duration of the project. Our objective is to explore how different teams adapted GitHub to their workflow and the impact of their GitHub behavior on the quality of their work.

[4] https://desktop.github.com/



(a) Assignment 1: SRS

(b) Assignment 2: SDD

(c) Assignment 3: STD

(d) Project Code

Figure 6.: Total number of commits per day for each assignment

5.1. Analyzing the Commit Timeline

We started our analysis by tracking the commit timeline of the different teams in the class. Specifically, for each assignment, we tracked the individual commit history (user name of the student who made the comment and the time of the comment) of each of the team members. The data was collected throughout the semester and was indexed in an Excel sheet. Extracting this information enabled us to measure the commit frequency per day for each individual assignment.

Our findings are presented in Fig. 6. This figure plots the number of commits made by all teams from the day the assignment was assigned to the day it was due. The results show that, for each assignment, the majority of commits happened toward the deadline. An exception to this pattern was the final code assignment, where the maximum number of commits happened five days prior to the deadline.

These results indicate the presence of academic procrastination, a phenomenon where students irrationally postpone, delay, or simply put off working on their assignments until the very last moment (Akerlof, 1991; Steel, 2007). This behavior is prevalent among undergraduate students. In fact, academic research has revealed that the overwhelming majority of college students were prone to procrastination (Knaus, 1973; Steel, 2007). In our analysis, academic procrastination can be clearly observed by looking at the commit timeline of individual assignments: the lion's share of submissions happened right before the deadline (Howell, Watson, Powell, & Buro, 2006; Rothblum, Solomon, & Murakami, 1986).

In general, our results have countered our earlier assumption that GitHub would help to reduce procrastination. Specifically, we assumed that GitHub's transparency would motivate our students to start working early by observing other students' contributions in their team as well as other teams in class. Unfortunately, the commit pattern provides evidence in favor of academic procrastination. The main takeaway message is that although GitHub provides a convenient method to track how and when students contribute to their assignments, it does not alter the student behavior in terms of assignment submission time. In other words, it is not a silver bullet for procrastination.

5.2. Analyzing the Number of Commits

Fig. 7 shows the average number of commits for all of the teams in class working on the different project assignments. The results show that, as the semester went on, students made fewer and fewer commits. This trend was obvious in the third assignment where the teams made an average of 24 commits in comparison to the first assignment (SRS), where they made an average of 43 commits. In general, most teams started to self-regulate after the first assignment, meaning that they became more efficient in committing as they figured out the risks of over-committing. Specifically, in their exit interview, 16 out of the 18 teams implied that at the beginning they were experimenting with committing and resolving merge conflicts. As the time went by, they figured out how to regulate the number of commits and minimize the number of merge conflicts.

Note that the number of commits for the code assignment was almost equivalent to the average number of commits for all other assignments. This can be explained based on the fact that the students were encouraged to start coding right after the SRS document was assigned.

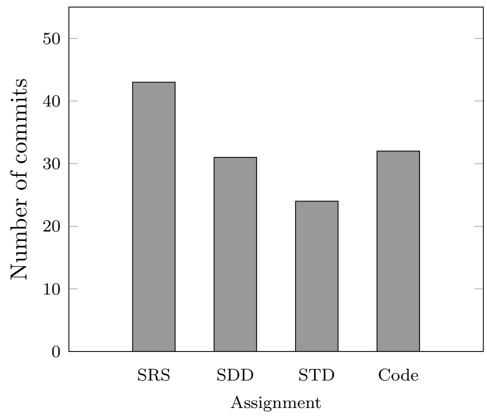


Figure 7.: The average number of commits for all teams per assignment

5.3. Team Organization

In this section, we explore how different teams utilized GitHub to assist with their team organization (**RQ₂**). To carry out our analysis, we tracked and analyzed the commit behavior of each individual student in each team. Our analysis shows that, in general, teams follow three different patterns:

- Equally committing: in three out of the 18 teams in our case study, every student in the team was committing to each assignment throughout the semester. Fig. 8-a shows a sample commit timeline for one of these teams. The timeline shows that almost all team members were equally active for all assignments. Our analysis shows that only these teams utilized branching as everyone in the team working concurrently on the project. Furthermore, our analysis of teams' performance shows that these teams did better than other teams in terms of the final project grade (average overall class grade is 90%). A plausible explanation is that these teams were technically balanced and everyone was well-motivated to participate in all aspects of the project, which was reflected in the commit pattern and the over all class grade.
- Designated configuration management engineer: in two out of the 18 teams, a team member was assigned the role of managing GitHub related tasks. Members of these teams would first work on the assignment through other mediums (Google docs or Dropbox), and then send the final copy to the configuration management engineer to submit to GitHub. Students in these teams indicated that they resorted to this approach to minimize merge conflicts. Fig. 8-c shows a sample commit timeline for this type of teams. The timeline shows that only one team member was actively committing. Further investigation revealed that these teams tend to follow a chief programmer structure. Under this structure, only one person is responsible for leading the entire project, while others tend to do less work. Our analysis of team performance shows that these teams tend to underachieve in comparison to other, more balanced teams (average overall class grade is 72%). A possible explanation is that these teams were not as technically-balanced to begin with, leading to the emergence of cowboy programmers and free-riders. This behavior was reflected in the commit pattern of the team.
- Experience-based committing: in 13 out of the 18 teams, the commit pattern of individual members followed the internal work assignment of the team. Specifi-

cally, such teams split the work along different lines: some teams split the work into coding and documentation, while others split the work based on assignments (SRS, SDD, STD) or specific implementation components (back-end, front-end). Fig. 8-b shows a sample commit timeline for one of these teams. The timeline shows that some students were active only at specific times (e.g., when the assignment that was assigned to them by the team was due). Students in these teams reported that they resorted to this structure to work around the variant levels of experience in the team also to minimize conflicts. According one student *"This style choice allowed for team members to divide project tasks into their own personal area of expertise without the possibility of managerial conflict"*.

5.4. Number of Commits vs. Performance

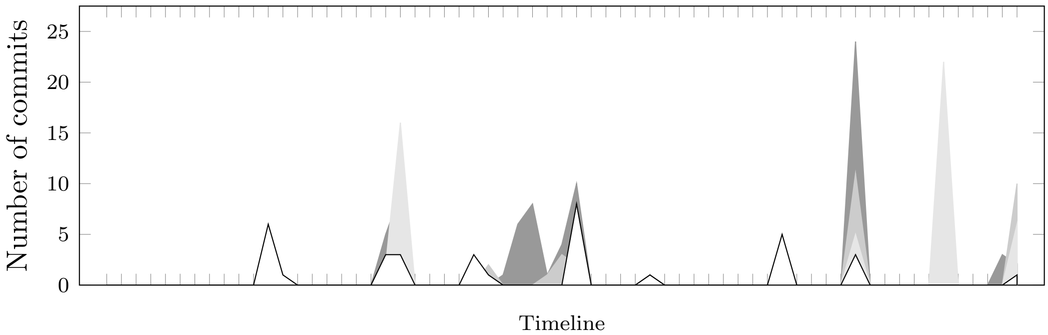
Under this part of our analysis, we explore the relation between the students' grades and their GitHub activity ($\mathbf{RQ_3}$). In CS 0000, a one-grade-fits-all strategy is typically used to grade team projects (Hayes et al., 2003). Specifically, all team members are assigned the same grade for each assignment. This grading strategy was adopted to deal with the large number of students in class. In particular, given the relatively large number of students and projects, there is no practical or objective way to distill the contributions of individual students to the project. In an ideal world, a one-grade-fits-all strategy should enhance the sense of responsibility among students toward their teams as students are aware that their behavior as individuals might affect the overall grade of the team. On the flip side, this grading strategy might encourage free-riders.

During the semester, all assignments were graded by the same TA and the class instructor according to a pre-defined rubric that had been used in the other two sections of CS 0000 taught previously by the same instructor. This rubric does not take GitHub into account. The correlation between the number of commits and the team grade is shown in Fig. 9. In general, the results show that, even though the relationship is slightly positive, no significant correlation is detected between the grade and the number of commits in any of the assignments.

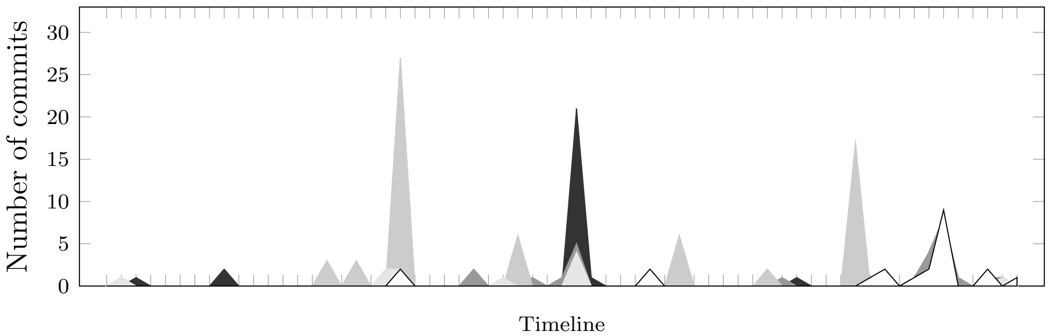
5.5. Quality of Commit Messages vs. Performance

In addition to analyzing the number of commits, the quality of commit messages (comments) was analyzed. Specifically, we downloaded and qualitatively assessed the messages made by each team throughout the period of the project. For each team, the authors went through the list of messages, classifying the teams based on the quality of their messages into three distinct categories:

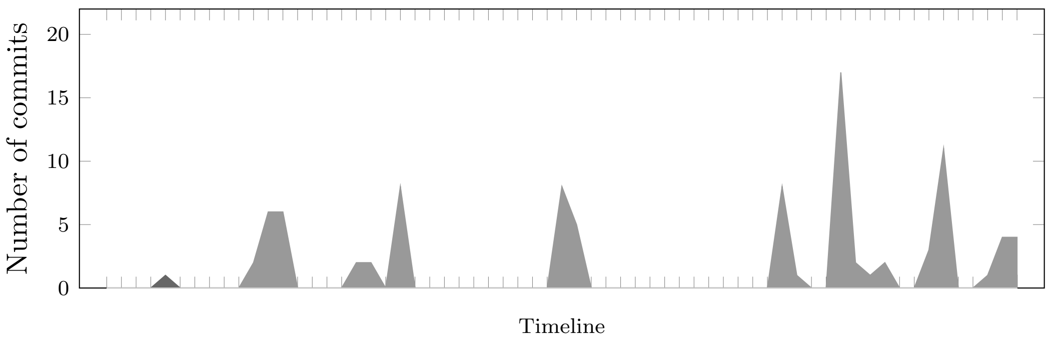
- Poor: teams with poor comment messages either did not comment at all, or left uninformative messages, such as *"John"* or *"something!."* Out of the 18 teams, nine were classified as poor.
- Average: teams with average quality messages left messages that we perceived as somewhat informative. In general, these comments acknowledged the specific changes made to the project, sometimes with the location of the change or the name of the person who committed, for examples, *"SRS changed by John"* and *"added section 2.5."* Out of the 18 teams, four were classified as average.
- Good: this category included teams that consistently left high quality messages that included meaningful descriptions of their changes, including what specific



(a) Equally committing members



(b) Experience-based committing



(c) A designated CM engineer

Figure 8.: Sample commit timelines for 3 different team organization styles. x-axes is time and y-axes is the number of commits. Different colors in the graph indicate different students in the team

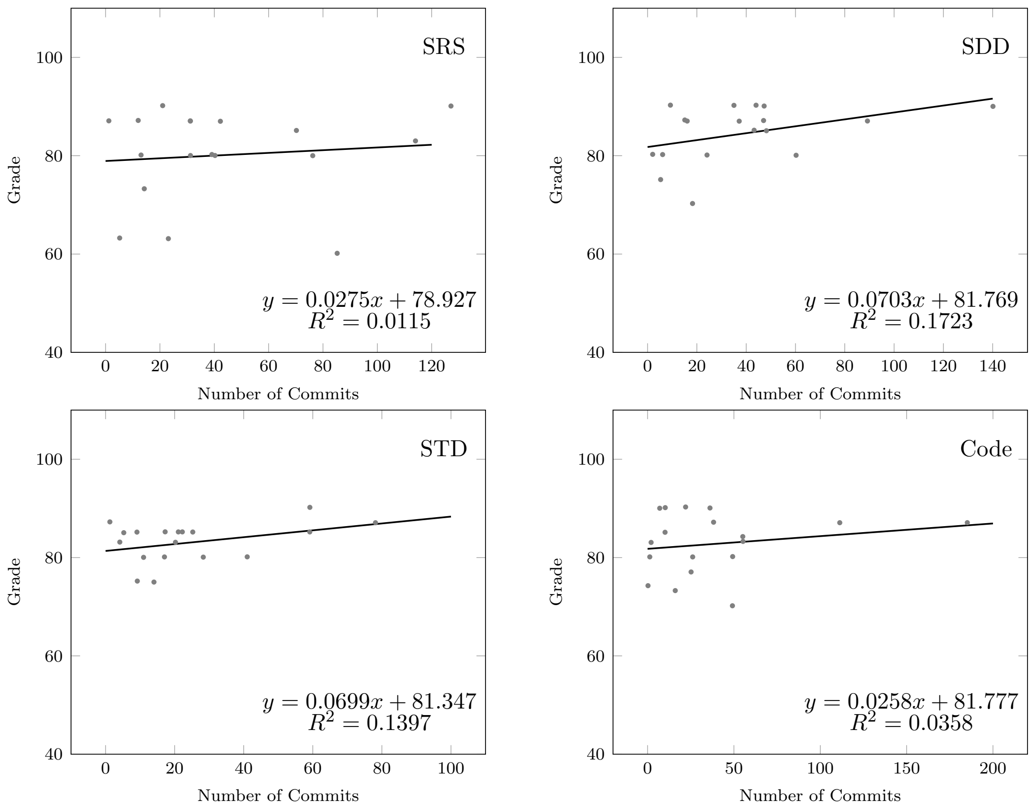


Figure 9.: Relationship between the total number of commits a team made and the grade they received, per assignment.

change was made and what files were affected. For example, *"Trent: added logical view diagram and minor edits to the GUI in page 5"* and *"update the bolding and spacing in section 3.6 of the SRS document."* Out of our 18 teams, five were classified as good.

We performed correlation analysis between the quality of the messages for each team (1 = poor, 2 = average, and 3 = good) and their final project grade. The correlation diagram in Fig. 10 shows that, even though the overall relation was positive, the correlation was not statistically significant. These results show that relying on the quality of commit messages for grading can be risky ($\mathbf{RQ_3}$). In general, our findings confirm what the students have expressed during their exit interview, the students did not feel the need to use the networking features of GitHub as they were in the same geographical location. They mainly informed each other about the changes they did by other means of communication such as short text messages or social networking apps such as GroupMe ($\mathbf{RQ_2}$).

6. Discussion and Recommendations

The tremendous success of the OSS movement in the past decade can be largely attributed the emergence of online version control systems, such as *GitHub* and *SourceForge* (Zambelich, 2015). These platforms have changed the way software is being produced and consumed, allowing more developers worldwide to join this movement

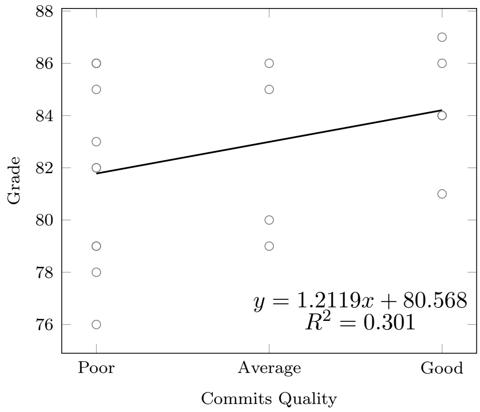


Figure 10.: Commits Quality vs. Grade

and share their experiences and knowledge in a nontraditional way.

Intensive research in this domain has revealed that OSS has the capacity to contribute directly to improved competitiveness, higher quality products, and lower costs of commercial software (Dinkelacker, Garg, Miller, & Nelson, 2002). Corporations have realized that they can benefit significantly from bringing selected open source best practices in-house. Therefore, having a basic knowledge of these platforms have become an essential skill that CS graduates should possess (Steinmacher, Silva, Gerosa, & Redmiles, 2015). Unfortunately, it is uncommon for CS departments to offer a full class on Configuration Management at the undergraduate level, and sufficiently covering this topic in a general SE class can be very challenging, especially when there is no designated lab for the class, as in the case of CS 0000. These observations expose an urgent need for research dedicated to identifying the best educational strategies for integrating such an unconventional software production paradigm into CS curricula. The case study reported in this paper is an attempt toward this goal. Specifically, our analysis in this paper has revealed several trends on how students would use such technology in the classroom. In what follows, we summarize these trends along with our recommendations to other instructors interested in using GitHub, or any other CM tool, in classroom.

- Enforcing GitHub did not impact the quality of students' work. In general, teachers should not be worried about the overhead of adapting such unconventional project management platform on the quality of students' submissions ($\mathbf{RQ_1}$ and $\mathbf{RQ_1}$).
- GitHub is not a silver bullet for preventing academic procrastination. Students are going to procrastinate regardless of the platform being used for project management and assignment preparation and submission. Overall, the students did not feel the need to start submitting earlier despite being encouraged otherwise ($\mathbf{RQ_1}$ and $\mathbf{RQ_2}$). Strategies such as micro-deadlines, where an assignment is sliced into multiple micro tasks that should be submitted separately, might help to mitigate this problem.
- Different teams integrated GitHub to their workflow differently. In a few teams, each member in the team committed to all assignments. The majority of teams followed a commit pattern based on the different expertise within the team. For example, team members who are good at coding tend to commit to the code assignment, while other team members commit to other documentation

assignments. In a few teams, only one team member was assigned to commit for the team.

- Students in balanced teams scored, on average, the highest class grades (projects and exams), while students in teams with a designated CM engineer generally scored the lowest in class. Such information can be used by instructors to assess team performance early in the process. Specifically, having a designated CM engineer could be a sign of a dysfunctional group (e.g., cowboy programmers and free riders). Usually such phenomena do not surface till the end of the semester, when students from these teams come forward with their problems. However, using GitHub, such behavior can be detected early based on the commit pattern (one team member committing). If detected, teachers can interfere early in the semester to give these groups more attention, resolve any personal conflicts among team members, or dissolve the team altogether. A suggestion to balance out such unbalanced teams is to find methods for encouraging students to participate. Recent research has revealed that including industrial partners in SE capstone projects can encourage students to learn new technologies and participate more actively in project activities Paasivaara, Vodă, Heikkilä, Vanhanen, and Lassenius (2018).
- There is no correlation between students' grades (quality of submissions) and the number of commits. This finding suggests that teachers should be careful before using number of commits, pull requests, and comments as a basis for grading. Furthermore, as mentioned in the previous point, different teams adapt GitHub to their workflow differently. This might result in undermining the progress of some teams, or students, who do not commit frequently. Overall, we conclude the level of GitHub's activity, as measured by the number of commits, is not a reliable proxy of team or individual student effort ($\mathbf{RQ_3}$). An alternative, perhaps more objective strategy would be to analyze the quality of students' commits or the number of added/deleted lines of code. This strategy might work in smaller classrooms, or when the entire class is working on the exact same project. This way the instructor can evaluate students' contributions to specific parts of the project more objectively (e.g., a section in a document or a feature in the code). However, performing such evaluation in large classrooms can be a tedious job, especially if students are working on different projects using multiple programming languages and utilizing a very broad range of external libraries. This effort can be reduced by using automated tools (e.g., Expertiza and TravisBuddy Bots) (Hu & Gehringer, 2019).
- There seems to be a consensus among students that conventional text editing tools, such as Google Docs or MS Word, can be more convenient for collaborating on documentation assignments (SRS, STD, and SDD). In fact, nine out of 18 teams mentioned that they first drafted their assignment documents using Google Docs and then converted them to MarkDown. Based on these observations, we recommend limiting GitHub usage to code submission only. Other assignments could be submitted using more traditional formats such as MS Word or PDF ($\mathbf{RQ_1}$).
- Our students have reported difficulties at the beginning of the class getting used to, or even understand, the various features of GitHub. However, the majority of the concerns were focused on dealing with merge conflicts. Therefore, we recommend a separate tutorial, or a pre-project assignment or class exercise, focusing on these problems early in the semester to speed up the learning process ($\mathbf{RQ_1}$).

- Students did not feel the need to utilize the social features of GitHub in their communication. The majority of the students reported that they used other tools to exchange information about the project. Among these tools, the GroupMe mobile application was the most popular platform. In general, most students reported that they did not feel comfortable communicating publicly, or even for their internal project discussions to be seen by the class TA or the instructor. Our students also reported that the features that the app provided (e.g., sharing images, videos and other media contents and getting notifications) made it more appealing than GitHub. In general, in classroom settings, where students are in the same geographical space, and likely to be in the same social circle, GitHub social features seem to be unnecessary for communication (**RQ2**).

7. Threats to Validity

The analysis presented in the paper takes the form of a case study. In empirical software engineering research, case studies are conducted to investigate a single entity or phenomenon in its real-life context within a specific time space (Wohlin et al., 2012). While case studies can be more realistic and easier to plan than experiments, they often suffer from several validity threats due to the lack of formalism that is typically associated with other forms of research methods, such as controlled experiments. In this section, we outline the main threats that could potentially undermine the validity of our results.

7.1. Internal Validity

Internal validity refers to the confounding factors that might affect the causal relations established in the experiment (Wohlin et al., 2012). Our case study was conducted in a classroom with 91 students participating as experimental subjects and the class instructor as the lead researcher. Given these settings, an internal validity threat might stem from the power dynamics in the classroom. Specifically, students might alter their responses in order to please the instructor and the TA, for example, claiming to have more programming experience or that using GitHub positively impacted their learning experience. To mitigate such threats, we enforced anonymity in both of our surveys. Furthermore, the primary class instructor was not present during the surveys. The students were also assured that participating in the survey and their answers did not have an impact on their grades.

Another threat to the internal validity might result from the fact that one of the main co-authors of the paper is the main instructor of the class. This might lead to confirmation bias, where experimenters would interpret evidence in ways that are partial to existing beliefs. However, as mentioned earlier, all assignments were graded by the instructor and the TA according to a predefined rubric that had been previously used in the two other sections of CS 0000 taught by the instructor.

To prevent the possibility of any other confounding effects, the students were assured multiple times that their level of activity on GitHub (number of commits, comments, and pull requests) did not impact their project grade by any means. This design decision was necessary to prevent altering the behavior of students (e.g., excessive committing to give the impression of a higher activity).

7.2. External Validity

Threats to external validity are conditions that limit the ability to generalize the results of the experiment (Wohlin et al., 2012). Case studies are inherently subjective and susceptible to selection bias. Therefore, we cannot fully ensure the generalizability of our findings. For instance, repeating our study under various settings, such as a different grading scheme, class size, or project specifications, might lead to different results. Nonetheless, given that our study was specifically aimed at computer science students, and the fact that the sample size was reasonable (91 subjects), a generalization over the whole population of computer science students is still possible. We still, however, acknowledge the fact that further formal experimentation is necessary to support our generalizability claims.

7.3. Construct Validity

Construct validity is the degree to which the various performance measures accurately capture the concepts they purport to measure (Wohlin et al., 2012). Construct validity is the degree to which the various performance measures accurately capture the concepts they purport to measure (Wohlin et al., 2012). For instance, we relied on students' assessment of their own skills as an indicator of their GitHub experience before and after the treatment. While such subjective methods are commonly used in the literature, especially when the study population consists of students (Haaranen & Lehtinen, 2015; Hsing & Gennarelli, 2019), we do acknowledge the fact that self-reporting might not be the best way to assess the shift in students' experience. Therefore, our findings regarding this matter, despite being significant, should be interpreted with care.

Another threat might stem from using students' grades as a proxy for judging the quality of their work. To mitigate this threat, students' assignments were graded based on a predefined rubric. Furthermore, assignments were graded without taking GitHub activity into account. In fact, to prevent any experimenter bias (e.g., the researcher would assign teams who showed more GitHub activity higher grades), the researcher who was responsible for collecting GitHub data did not participate in the grading process.

7.4. Conclusion Validity

Threats to the conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment (Wohlin et al., 2012). In our case study, we compared students' experience in GitHub before and after the class using the non-parametric Wilcoxon Rank Sum test. One of the assumptions of this test is to have matched pairs (before and after measurements for each subject). Despite this assumption, Wilcoxon Sign-Rank test can still be used within the matched-pairs study design, even if establishing the pairs is not possible. In our study, establishing pairs for the entry and exit surveys was not possible due to anonymity of the surveys. However, the same sample was used for both surveys, which makes Wilcoxon Sign-Rank test applicable to our study design.

8. Conclusions and Future Work

This paper reported the results of a case study on using GitHub as a collaboration platform for student team projects in a Software Engineering class.

The objective of our case study was to explore the different ways students utilized such a platform to manage their projects and assignments. The research method consisted of an entry and exit surveys, an exit interview, and a qualitative analysis of the commit behavior of students in class. The results showed that, despite the steep learning curve, enforcing GitHub in SE team projects can be an effective way for students to learn about online version control platforms without impacting the quality of their work. However, the level of GitHub activity (e.g., number of commits of quality of commit messages) cannot be used as a reliable proxy for assessing individual student effort. In particular, students in different project teams utilized GitHub differently. Therefore, it was not possible to distill a single metric for work assessment that would work for the entire class. Furthermore, the results revealed that the social features of GitHub were of limited use to the students as they felt more comfortable communicating using other, more convenient, tools.

The case study reported in this paper contributes to the existing research effort on developing effective educational strategies for integrating unconventional software production and collaboration platforms into existing CS curricula. Our future work will include conducting other case studies, using more recent versions of open source tools, and targeting other variables that are often associated with online collaborative platforms, such as gender bias, team structure, and project type. Our long term goal is to derive a formal theory that can explain the different factors that control student collaboration in software-intensive team projects, predict student academic behavior when using collaborative platforms (e.g., procrastination, free riders, and cowboy programmers), and aid in individual as well as team student effort assessment.

References

Akerlof, G. (1991). Procrastination and obedience. *The American Economic Review*, *81*(2), 1–19.

Buchta, J., Petrenko, M., Poshyvanyk, D., & Rajlich, V. (2006). Teaching evolution of open-source projects in software engineering courses. In *Ieee international conference on software maintenance* (pp. 136–144).

Buffardi, K. (2015). Localized open source collaboration in software engineering education. In *Ieee frontiers in education conference* (pp. 1–5).

Carver, J., Jaccheri, L., Morasca, S., & Shull, F. (2004). Issues in using students in empirical studies in software engineering education. In *Software metrics symposium* (pp. 239–249).

Carver, J., Jaccheri, L., Morasca, S., & Shull, F. (2010). A checklist for integrating student empirical studies with research and teaching goals. *Empirical Software Engineering*, *15*(1), 35–59.

Chao, J. (2007). Student project collaboration using wikis. In *Conference on software engineering education & training* (pp. 255–261).

Ciolkowski, M., Laitenberger, O., Vegas, S., & Biffl, S. (2003). Practical experiences in the design and conduct of surveys in empirical software engineering. *Empirical methods and studies in software engineering*, 104–128.

Clark, N. (2005). Evaluating student teams developing unique industry projects. In *Australasian conference on computing education* (pp. 21–30).

Clark, N., Davies, P., & Skeers, R. (2005). Self and peer assessment in software engineering

projects. In *Australasian conference on computing education* (pp. 91–100).
Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Stafford, J., ... Nord, R. (2000). Documenting software architectures: Views and beyond. Addison-Wesley.
Colbeck, C., Campbell, S., & Bjorklund, S. (2000). Grouping in the dark: What college students learn from group projects. *The Journal of Higher Education*, *71*(1), 60–83.
Coppit, D. (2006). Implementing large projects in software engineering courses. *Computer Science Education*, *16*(1), 53–73.
Coppit, D., & Haddox-Schatz, J. (2005). Large team projects in software engineering courses. In *Acm sigcse bulletin* (pp. 137–141).
Curtis, T. (2001). So you wanna be a cowboy. *IEEE Software*, *18*(2), 112–111.
Dinkelacker, J., Garg, P., Miller, R., & Nelson, D. (2002). Progressive open source. In *International conference on software engineering* (pp. 177–184).
Feliciano, J., Storey, M.-A., & Zagalsky, A. (2016). Student experiences using github in software engineering courses: A case study. In *International conference on software engineering companion* (pp. 422–431).
Gates, A., Delgado, N., & Mondragón, O. (2000). A structured approach for managing a practical software engineering course. In *Annual frontiers in education conference. building on a century of progress in engineering education* (pp. 21–26).
Gokhale, A. (1995). Collaborative learning enhances critical thinking. *Journal of Technology Education*, *7*(1), 22–30.
Goold, A., Augar, N., & Farmer, J. (2006). Learning in virtual teams: Exploring the student experience. *Journal of Information Technology Education*, *5*, 477–490.
Haaranen, L., & Lehtinen, T. (2015). Teaching Git on the side: Version control system as a course platform. In *Acm conference on innovation and technology in computer science education* (pp. 87–92).
Hansen, R. (2006). Benefits and problems with student teams: Suggestions for improving team projects. *Journal of Education for Business*, *82*(1), 11–19.
Hayes, J., Lethbridge, T., & Port, D. (2003). Evaluating individual contribution toward group software engineering projects. In *International conference on software engineering* (pp. 622–627).
Heckman, S., & King, J. (2018). Developing software engineering skills using real tools for automated grading. In *Acm technical symposium on computer science education* (pp. 794–799).
Hollar, A. B. (2006). Cowboy: An agile programming methodology for a solo programmer.
Howell, A., Watson, D., Powell, R., & Buro, K. (2006). Academic procrastination: The pattern and correlates of behavioural postponement. *Personality and Individual Differences*, *40*(8), 1519–1530.
Hsing, C., & Gennarelli, V. (2019). Using github in the classroom predicts student learning outcomes and classroom experiences: Findings from a survey of students and teachers. In *Acm technical symposium on computer science education* (pp. 672–678).
Hu, Z., & Gehringer, E. (2019). Use bots to improve github pull-request feedback. In *Acm technical symposium on computer science education* (pp. 1262–1263).
Johansson, C. (2000). *Communicating, measuring and preserving knowledge in software development* (Unpublished doctoral dissertation). Ronneby, Sweden.
Kelleher, J. (2014). Employing Git in the classroom. In *World congress on computer applications and information systems* (pp. 1–4).
Kertész, C.-Z. (2015). Using github in the classroom-a collaborative learning experience. In *International symposium for design and technology in electronic packaging* (pp. 381–386).
King, P., & Behnke, R. (2005). Problems associated with evaluating student performance in groups. *College Teaching*, *53*(2), 57–61.
Klassen, R., Krawchuk, L., & Rajani, S. (2008). Academic procrastination of undergraduates: Low self-efficacy to self-regulate predicts higher levels of procrastination. *Contemporary Educational Psychology*, *33*(4), 915–931.
Knaus, W. (1973). Overcoming procrastination. *Rational Living*.

Liu, C. (2005). Using issue tracking tools to facilitate student learning of communication skills in software engineering courses. In *Conference on software engineering education & training* (pp. 61–68).

Lo, D., Nagappan, N., & Zimmermann, T. (2015). How practitioners perceive the relevance of software engineering research. In *Joint meeting on foundations of software engineering* (pp. 415–425).

Manotas, I., Bird, C., Zhang, R., Shepherd, D., Jaspan, C., Sadowski, C., ... Clause, J. (2016). An empirical study of practitioners' perspectives on green software engineering. In *International conference on software engineering* (p. 237-248).

Ohlsson, L., & Johansson, C. (1995). A practice driven approach to software engineering education. *IEEE Transactions on Education*, *38*(3), 291–295.

Paasivaara, M., Vodă, D., Heikkilä, V. T., Vanhanen, J., & Lassenius, C. (2018). How does participating in a capstone project with industrial customers affect student attitudes? In *International conference on software engineering: Software engineering education and training* (pp. 49–57).

Punter, T., Ciolkowski, M., Freimut, B., & John, I. (2003). Conducting on-line surveys in software engineering. In *International symposium on empirical software engineering* (pp. 80–88).

Reisbig, A., Jr, M. H., White, M., & Rush, B. (2007). Improving response rates: introducing an anonymous longitudinal survey research protocol for veterinary medical students. *Journal of Veterinary Medical Education*, *34*(2), 194–201.

Rothblum, E., Solomon, L., & Murakami, J. (1986). Affective, cognitive, and behavioral differences between high and low procrastinators. *Journal of Counseling Psychology*, *33*(4), 387.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131.

Seppälä, O., Auvinen, T., Karavirta, V., Vihavainen, A., & Ihantola, P. (2016). What communication tools students use in software projects and how do different tools suit different parts of project work? In *International conference on software engineering companion* (pp. 432–435).

Slavin, R. (1980). Cooperative learning. *Review of Educational Research*, *2*(50), 315-342.

Steel, P. (2007). The nature of procrastination: A meta-analytic and theoretical review of quintessential self-regulatory failure. *Psychological Bulletin*, *133*(1), 65.

Steinmacher, I., Silva, M., Gerosa, M., & Redmiles, D. (2015). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, *59*, 67–85.

Wilkins, D., & Lawhead, P. (2000). Evaluating individuals in team projects. In *Sigcse technical symposium on computer science* (p. 172-175).

Williams, D., Beard, J., & Rymer, J. (1991). Team projects: Achieving their full potential. *Journal of Marketing Education*, *13*(2), 45–53.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslèn, A. (2012). Experimentation in software engineering. Springer.

Zagalsky, A., Feliciano, J., Margaret-AnneStorey, Zhao, Y., & Wang, W. (2015). The emergence of github as a collaborative platform for education. In *Acm conference on computer supported cooperative work & social computing* (pp. 1906–1917).

Zambelich, A. (2015). *How github conquered google, microsoft, and everyone else.* Retrieved August - 2018, from `https://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else/`