4-9-2018

Using GitHub in Large Software Engineering Classes: An Exploratory Case Study

Miroslav Tushev
Louisiana State University and Agricultural and Mechanical College

USING GITHUB IN LARGE SOFTWARE ENGINEERING CLASSES: AN EXPLORATORY CASE STUDY

A Thesis

Submitted to the Graduate Faculty of
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

Division of Computer Science and Engineering

by
Miroslav Tushev
Specialist Degree, The Russian Academy of National Economy and Public Administration, 2014
August 2018

To my wife and my parents.

Acknowledgments

I would like to thank all the people who helped make this possible. Specifically, I would like to acknowledge my major professor, Dr. Anas Mahmoud, who guided me and helped me tremendously with writing this work and putting it all together; my committee members: Dr. Doris Carver and Dr. Gerald Baumgartner, who agreed to be serve on my defense and to give their reviews of my work; and Grant Williams, PhD student, who helped my with data collection. Thank you.

Table of Contents

List of Tables

List of Figures

Abstract

GitHub has been recently used in Software Engineering (SE) classes to facilitate collaboration in student team projects. The underlying tenet is that the technical and social feature of GitHub can help students to communicate and collaborate more effectively as a team as well as help teachers to evaluate individual student contribution more objectively. To shed more light on this, in this case study, we explore the benefits and drawbacks of using GitHub in SE classes. Our study is conducted in a software engineering class of 91 students divided into 18 teams. Our research method includes an entry and an exit surveys and a qualitative analysis of students' commit behavior throughout the period of the project. Our findings show that *a)* enforcing GitHub in SE classes can be an effective approach for enhancing students' skills in configuration management and version control, and *b)* despite the steep learning curve, most teams managed to optimize their commit behavior over time. In terms of student evaluation, our analysis exposed the risks of using GitHub for individual effort assessment. The work in this paper provides several valuable insights for researchers and makes several recommendations for practitioners (teachers) about integrating GitHub in SE classes.

Chapter 1
Introduction

Software Engineering (SE) classes have become an essential part of the 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 enhance the students' ability to function in teams and communicate effectively with a broad range of audiences. The main objective of Software Engineering classes 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 mid-size group project that students have to work on during the class. The educational objectives of the project are to *a)* reinforce the concepts being taught in the classroom, and *b)* simulate a realistic, industrial software engineering environment [1, 2, 3]. However, due to the limited time frame (typically 4 months), large classroom size, and undergraduate students' general lack of real-life experience, such projects often pose many challenges for students and teachers, including problems with inter-group communication, collaboration, and evaluation of individual student contribution (effort) [4, 5, 3].

Recent research has revealed a communication problem for students working in teams [6]. This problem can be attributed to many factors. For instance, 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 use [7, 8, 9, 10].

Another major challenge facing student teams is inter-team collaboration [11, 12, 13]. Despite being encouraged otherwise, students often end up forming groups of people whom they are comfortable working with (mainly friends), rather than based on the technical merit [14]. This leads to the formation of unbalanced teams in terms of technical and soft skills. In most cases, unbalanced groups lead to the emergence of *cowboy programmers* [15, 16], where one dominant person in the group does all the work, while the rest struggles to maintain the same level of contribution, or loses interest in the entire project, converting to *free-riders* who are satisfied with not being active in their teams [17, 18, 19, 2].

From a teacher's perspective, a key challenge facing teachers in SE class team projects is how to objectively evaluate individual team members. This challenge stems from the fact that individual contributions are typically not separately quantifiable. To overcome this problem, several grading schemes have been proposed in the literature, including individual student grading and the one-grade-fits-all approach [2, 3, 20, 21]. In the former approach, peer evaluation, where students are asked to evaluate each others' performance, or self-evaluation, where students are asked to submit a report detailing their specific contributions to the project, are used. In the latter approach, one grade is assigned for the entire group based on their overall performance as a group. 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 student effort [22, 3, 23].

In an attempt to overcome these challenges, online version control systems have been recently utilized in SE classes as a teaching strategy to facilitate student collaboration in team projects. These platforms, such as *GitHub* and *SourceForge*, provide programmers with online hosting services to upload, share, and maintain their code, and establish and manage world-wide social networks of developers at unprecedented scales. These unique technical and social features of such platforms have made them an appealing tool to be

used as a means to facilitate student communication, collaboration, and evaluation in SE class team projects [24, 25].

Despite several early studies on utilizing GitHub in SE class projects [24, 25, 26], 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 el al. [27], *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 projects, the context is the CSC 4330 class offered by the Computer Science and Engineering Department at Louisiana State University, and our time-frame is the Fall semester of 2016. Our main objective is to explore the impact of using such a platform on student's individual and aggregate performance. Case studies can be powerful tools for establishing early knowledge in unexplored domains. Such knowledge can be aggregated and utilized for formulating and testing formal hypotheses, and eventually, building a unified theory for the domain. In particular, our specific contributions in this paper are:

- We conduct pre- and post-study surveys to measure the impact of enforcing GitHub in SE classes and outline the main challenges faced by students when using such a platform for collaboration and assignment submission.
- We make several recommendations for instructors about enforcing GitHub in SE classes and describe the risks associated with using this platform for student evaluation.
- 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 benefits and drawbacks of such an unconventional tool in educational settings.

The rest of the paper is organized as follows: Chapter 2 reviews seminal related work

and outlines our motivations. Chapter 3 outlines the setup of our case study. Chapter 4 describes the pre- and post-study surveys and analyzes their results. Chapter 5 discusses the results of our qualitative data analysis. Chapter 6 provides a discussion and recommendations for teachers. Chapter 7 describes the potential threats to our study validity. Finally, Chapter 8 concludes the paper and discusses prospects of future work.

Chapter 2
Related Work

This section reviews seminal work on student collaboration and evaluation in SE team projects, summarizes existing work on the utilization of GitHub in educational settings, and outlines our main motives in this paper.

2.1 Team Projects in SE Courses

Student collaboration and evaluation in SE class projects has 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 the gap between class and industrial practices, Buchta et al. [19] 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) developed for the class was used to coordinate teamwork. Such CVS would allow the students to collaborate over their projects, even though the students were not physically present in the same location. The students were asked to submit a report after the completion of each phase. 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 problems related to individual student accountability as well as increased student motivation and their understanding of the software engineering process.

Chao [10] 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 the students, Coppit and Haddox-Schatz [2] presented an approach to teaching a one-semester large-scale software engineering course in which students work 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 reported several challenges regarding the choice of the project as well as the integration of under- and overachieving students in large scale teams.

Hayes et al. [3] tackled the challenge of fairly and accurately discerning individual student efforts in SE team projects for evaluation purposes. Specifically, the authors presented and discussed several grading approaches and best practices for evaluating individual student contributions. The authors made several recommendations to ensure the fairness and consistency of the grading process. These recommendations included, for instance, allowing team members to evaluate each other, but to carefully and frequently monitor this process to prevent the mob mentality among students and to use project demonstrations or quizzes to further test their project knowledge.

Goold et al. [9] investigated the use of an online learning environment platform to enhance students' experience when working in virtual teams. Three anonymous student surveys were conducted to elicit feedback specific to student opinions about their experiences of working in virtual teams within the learning environment. The results showed that, across the three surveys, 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 group 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. [21] 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 own contribution as well as others' 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 GitHub as a Teaching Tool

Motivated by its undeniable positive impact on the OSS movement as well as its social and technical features, GitHub has been recently utilized in SE and programming classrooms as a tool for managing student projects. This phenomenon has encouraged researchers to further investigate the benefits and drawbacks associated with using GitHub in educational settings. For instance, in their study, Zagalsky et al. [24] examined how GitHub could improve or possibly hinder the educational experience for students and teachers. In particular, the authors conducted a qualitative study to understand how GitHub is being used in education, and the motivations, benefits and challenges it brings. The study consisted of analyzing online posts of personal experiences in using GitHub in classroom along with interviews with faculty who used GitHub to support teaching or learning. The analysis revealed that GitHub is 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. However, several limitations included barriers to entry, long learning curve, and lack of direct support for popular educational formats (e.g., PDF and LaTeX) were reported.

In a follow-up study, Feliciano et al. [25] 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 material dissemination, lab work submission, and student 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 unfamiliarity with Git and GitHub, and the general lack of educational features to support grading and assignment management.

Kertesz [26] carried out a learning experiment about using GitHub as a collaborative platform for students to do their homework and classroom assignments in an operating systems laboratory. The results of analyzing students commit patterns as well as an exit student survey indicated that, in general, students found GitHub to be useful in learning from each others' 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 who were not satisfied with GitHub.

Haaranen and Lehtinen [28] 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 on a large Web software development class utilizing Git. Data was collected using a mixed approach, combining a feedback survey collected after the 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.

2.3 Motivation and Research Questions

Our brief review of related work has revealed some patterns in the research on using GitHub in educational settings. First, most of the analysis takes the form of case studies, action research, or experience reports. 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 [27].

The assumptions behind utilizing GitHub in SE class projects fall under the tenets of the collaborative learning theory, which describes situations in which two or more people build synchronously and interactively a joint solution to a specific problem. This theory suggests that learning is inherently a social process, thus, emphasizing the extent and quality of the exchanges that occur within groups of students in collaborative environments as a way for increasing critical thinking and team spirit [16, 29]. 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 student collaboration and their sense of teamwork. Another objective of enforcing a tool such as GitHub in classrooms 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 used in industrial settings. However, by using GitHub as the main platform for managing their term project, students can get hands-on experience using such a platform in semi-professional settings.

In terms of limitations, our review shows that the most common challenges that limit the utilization of GitHub in SE classes include the steep learning curve often associated with introducing a new technology in classroom settings, the lack of features that can

support certain class tasks, such as assignment submission and grading, and the conflicts that might arise from the variation between students in their GitHub experience.

Our review also shows that multiple studies have examined using GitHib's tracking features as a basis for student evaluation [26, 30, 28]. 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 [28, 24, 30]. 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 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 [30].

In summary, our review shows that GitHub can potentially improve teaching and learning experience in SE classrooms. However, there is still a research gap on how such platform actually affects student behavior, especially in large classroom settings (60+ 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 classroom settings. To guide our analysis, we formulate the following research questions:

- **RQ1.** Does enforcing GitHub enhance students' configuration management skills?
- **RQ2.** What are the main benefits and limitations of using GitHub for code and assignment submission?
- **RQ3.** How do students utilize GitHub in SE classes' team projects?
- **RQ4.** Can GitHub be used as a basis for evaluation of individual effort?

Chapter 3
Case Study Setup

CSC-4330, the Software Systems Development class, is a core senior-level software engineering class offered by the Department of Computer Science and Engineering at Louisiana State University. The class has a significant semester-long mid-size software project component that students are expected to execute in order to pass. At the beginning of the semester, students are asked to form their project teams and choose their project. Each team should have between 4-5 students. The project is divided into 5 assignments. These assignments can be described as follows:

- Software Requirements Specification (SRS): in the first assignment, students are required to gather and document the main functional and non-functional specifications of their systems.
- Software Design Document (SDD): in the second assignment, students are required to design the main modular components of the system and their relations. This document also includes other system design aspects, such as the database and the hardware views of the system (if any).
- Software Test Document (STD): in this assignment, students are required to describe their test plan and design a set of test cases for their system.
- Code: at the end of the semester, students in CSC-4330 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 their meeting time, configuration management plan, commercialization plan, risk management plan, and finally, their individual project contributions.

At the end of the semester, each team has to present their final product. This presentation is typically held in a conference-like format in front of representatives from the

industry and academia as well as other students in the class. During the presentation, each team member has to present a part of the project and talk briefly about their specific contribution. Students are graded based on their level of professionalism, including showing up on time, their dress-code, presentation skills, and ability to answer audience questions.

In the Fall of 2016, the class had 88 students, divided into 18 project groups. 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.

The teaching assistant of the class held a tutorial to introduce students to GitHub at the beginning of the semester. The tutorial included introducing students to basic concepts of configuration management and version control, such as creating a repository, commits, pull requests, merge, forks, and branches. The tutorial also included instruction on how to create basic MarkDown documents, including tables and figures. The students were further encouraged to watch multiple YouTube GitHub tutorials that were recommended by the instructor. A very important point to be pointed out is that students were assured that their final project grade would not depend on their level of GitHub activity (i.e., number of commits). In other words, students were told that they were free to adopt whatever commit strategy they felt comfortable with as a team. Our main objective was to track how different groups of students would utilize such a platform in the absence of an evaluation component.

The students were given the freedom to chose their projects and whatever technologies and tools they wanted to work with. To ensure that all teams had projects of a decent size, no video-games or single feature apps were allowed. Students had to present their ideas at the beginning of the semester for the teacher and the TA to approve. The structure of

the team was left for the students to decide. The students were introduced into several team formations, including ego-less (all team members share equal responsibilities), chief-programmer, and hierarchical structures. Unfortunately, there was no designated lab for the class. The students were expected to meet outside of the class to organize their teams.

The research methods used in our study included a pre-study survey and exit survey and a qualitative analysis of students' commit patterns. These methods are commonly used to collect data in case-study research [27]. In what follows, we describe each of these methods along with our main findings.

Chapter 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 [31]. In general, surveys can take the form of direct interviews or written questionnaires. While interviews can help to elicit more thorough and more honest responses from subjects, it is often expensive to conduct interviews at a large scale. Questionnaires, on the other hand, can be cheaper to execute, especially when the population is so large that a face-to-face interview would be infeasible, such as opinion polls and market research. In software engineering research, surveys have become a standard tool for data collection [32]. Interview and questionnaire surveys are frequently conducted to gather rapid feedback from software engineering practitioners on a variety of topics (e.g., [33, 34]). Furthermore, surveys are commonly used in classroom research to elicit students' feedback toward new teaching strategies [10, 19].

In our 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 [35, 36].

Our study included two anonymous surveys: a descriptive pre-study survey and an exploratory post-study survey. The pre-study survey was used to collect general descriptive information about the population (students in the Fall of 2016's CSC-4330 class). The post-study (exit) survey was used to explore how the applied treatment (i.e. enforcing GitHub in the class) influenced students' experience [37]. In what follows, we describe these surveys and their results in greater detail.

Table 4.1: A questionnaire sample

Question	**Answer Variants**
What is your gender?	Male / Female
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)

4.1 Survey Design

The pre-study 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 questionnaire 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 4.1. The response rate was 100% (i.e., all of the students in the class completed the survey). The survey results were transcribed and coded in *Microsoft Excel*, and then analyzed in *IBM SPSS* statistical package.

The post-study survey contained the same questions as the pre-study survey and an additional open-ended question to collect the students' perceptions of using GitHub in a

SE course (*"Did you face any challenges using Github for this class?"*). The post-survey was completed by 84 students: 45 juniors, 38 seniors, and 1 sophomore. The response rate was again 100% (7 students had dropped the class). The results were also coded in the same manner and analyzed in *IBM SPSS*. In what follows, we describe the main outcomes of the surveys.

4.2 Pre-Survey Results Analysis

Table 4.2 presents the results of the pre-survey. The results show that student cohort was split almost equally between 3rd (junior) and 4th-year (senior) students. As expected, juniors are less experienced with programming and know fewer programming languages than seniors. Surprisingly, seniors have reported almost the same level of experience in GitHub as juniors. In terms of gender, female students reported less experience in GitHub and less years of programming experience and knew fewer programming languages than male students.

Table 4.2: Descriptive statistics for the pre-survey: the number of programming languages, experience on GitHub, and years of programming experience, grouped by gender and year of education

		Prog Languages				**Exp. on GH**				**Years of Exp.**			
	N	Mean	Med.	StD	Range	Mean	Med.	StD	Range	Mean	Med.	StD	Range
Male	78	4.19	4.00	2.01	(1-9)	2.01	2.00	0.81	(1-4)	3.78	3.00	2.14	(1-12)
Female	13	3.31	3.00	1.38	(2-6)	1.54	2.00	0.52	(1-2)	2.58	2.50	0.67	(2-4)
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)

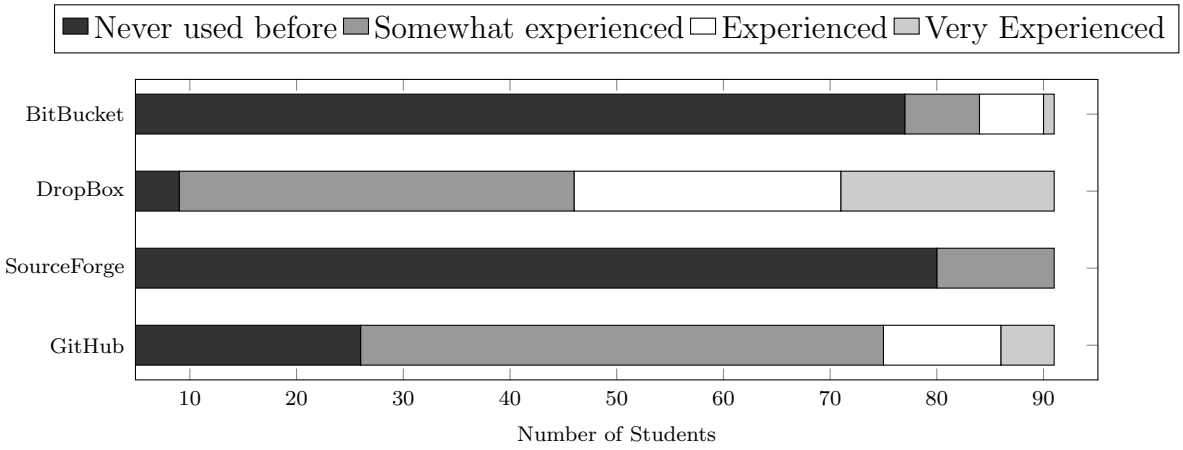
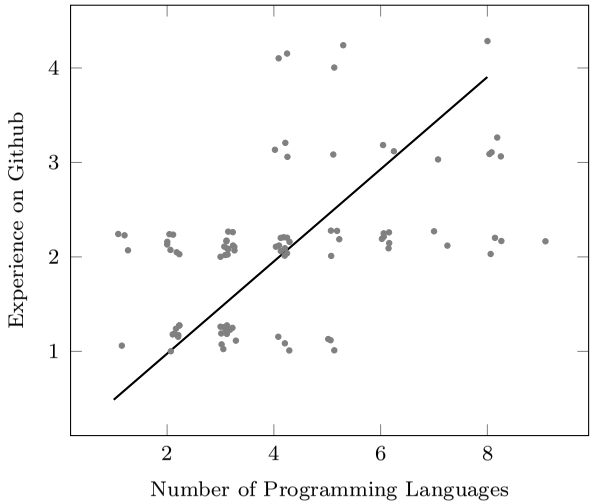
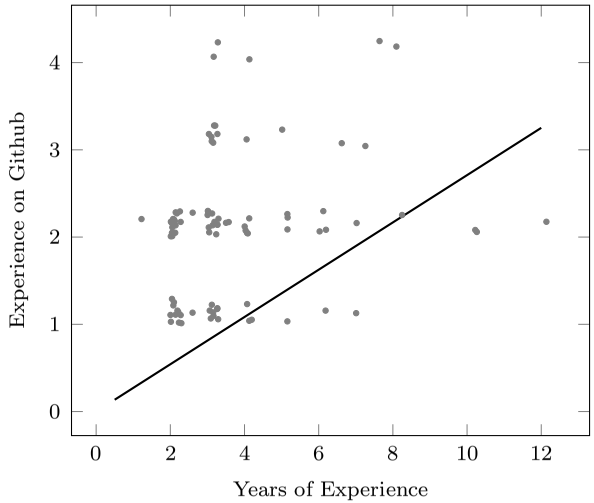


Figure 4.1: Students' experience with GitHub and other platforms

A correlation analysis was conducted to reveal any relationship between the number of programming languages and experience on GitHub and the years of general software development experience and specific experience on GitHub. The results, in Figure 4.2a, show the correlation graph between number of programming languages and experience on GitHub. In general, the more programming languages a student knows - the more experienced on GitHub he or she is. Spearman's correlation reports a strong positive statistically significant relationship between these two variables ($p = 0.000$). Figure 4.2b shows the correlation graph between the years of programming experience and experience on GitHub. Spearman's correlation also shows statistical significance ($p = 0.010$) between these two indicators. A more experienced student in programming is more likely to be more experienced on GitHub.



(a) Number of Programming Languages and Experience
on GitHub, ($p = 0.000$)



(b) Years of Experience and Experience on GitHub, ($p = 0.010$)

Figure 4.2: Spearman's correlation for Number of Programming Languages,Years of Experience and Experience on GitHub. Noise was added for overlapping points.

Figure 4.1 shows a breakdown of students' experience with GitHub, SourceForge, Dropbox, and BitBucket. Most of the students reported a very good experience with Dropbox, followed by GitHub. Around a half of the students indicated that they were somewhat experienced with GitHub and a forth of the students reported no knowledge of GitHub before. The majority of the students pointed out that they had never used SourceForge or BitBucket before. In addition, several students had reported some sort of experience with OneDrive and Google Drive, as well other platforms, such as SVN, TFS, and Visual Studio Online.

4.3 Post-Study Survey Analysis

To answer **RQ1**, we compared students' responses for GitHub experience in the pre-survey with that of the post-survey to measure the impact of enforcing GitHub on the students' experience in the platform. The plots in Figure 4.3 demonstrate the GitHub experience as reported by the students in the pre-survey and post-survey. In the pre-survey, a small number of students identified themselves 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 post-survey, the number of *"Somewhat experienced"* decreased, but the number of more experienced students increased drastically, especially in the *"Experienced"* category. The mean for the post-survey increased, settling between *"Somewhat experienced"* and *"Experienced"*. The conventional approach to test for the statistical significance would suggest using a dependent (pair-wise) t-test. However, due to anonymity of the surveys, we can not match the students in the pre- and the post-surveys. Therefore, an independent t-test was used. This test is appropriate to analyze our data, despite of the fact that the data violate the assumption of independence of observations (i.e. the samples are not independent). The results showed the statistically significant difference in GitHub experience for all students ($t = -5.144, p < 0.001$), females ($t = -3.922, p < 0.001$), and males ($t = -4.392, p < 0.001$).

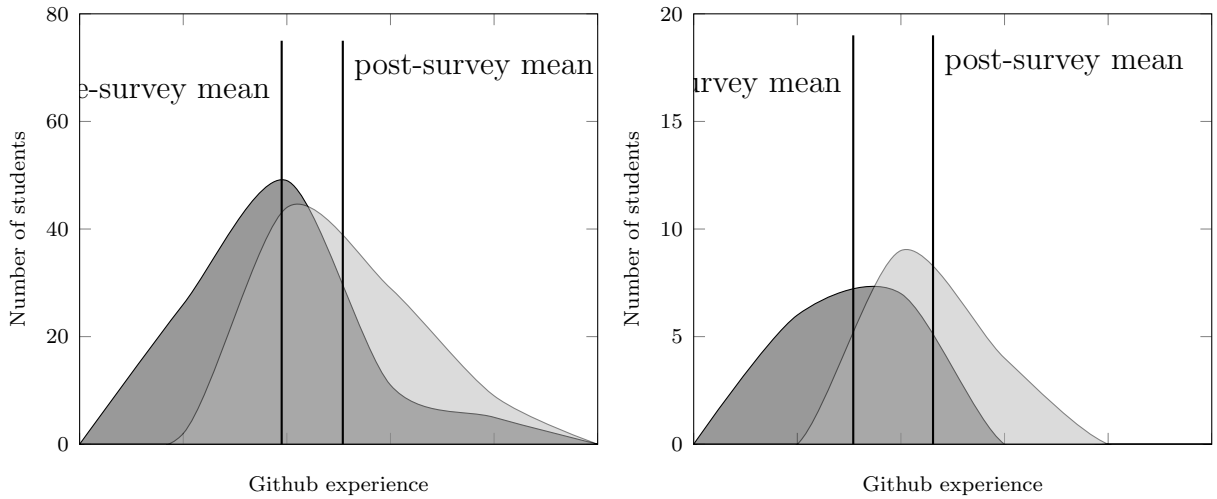
The last question in the post-survey asked students to share the challenges they faced

using GitHub during the semester. Students answers' were analyzed and presented in Table 4.3. Out of 84 students, 44 mentioned that they did not experience any difficulties using GitHub. The difficulties (**RQ2**) that were commonly reported by the students could be described as follows:

- **Resolving merge conflicts**: several students have 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 be 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 has 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 others have reported in the literature, the steep learning curve was an issue for some students. These are mainly the students who have never worked with GitHub, or any online version control systems, before the CSC-4330. This issue often arises from the perceived complexity of such platforms. A student writes: *"Bit of learning curve when I first started...was a bit complicated."* and *"GitHub is too hard to understand and use"*.
- **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: *"...2 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) have reported difficulties when dealing with MarkDown, especially when formatting with figures and tables. A student writes: *"I faced a few issues with formatting in MarkDown..."*. Consequently, some

Table 4.3: A summary of challenges of using GitHub mentioned by students in the post-survey

Challenges	**Count**
NA	44
Resolving merge conflicts	9
Learning curve	6
Working on the same file	5
Markdown	4
Branching	3
Other	10



(a) All students (b) Female students

Figure 4.3: Frequency distribution comparison of GitHub experience in the pre- and the post-surveys

students have suggested using other, easier to format, platforms such as Google Documents: *"I feel like google docs is a better option." "...there are simply better options for the document submissions."*

Chapter 5
Analyzing Student Commit Behavior

In this section, we explore the impact of using GitHub on student collaboration and submission behavior. In particular, we aggregate and analyze students' committing patterns over the project assignment to understand how different groups adapted GitHub to their projects.

To determine if the overhead of this steep learning curve has impacted the quality of students' work, we compare the students grades from the class with the average grades from two other sections of CSC 4330 where GitHub was not used. The results in Figure 5.1[1] provide an 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 T.A. and using the exact same predefined rubric. Therefore, these concerns were minimized.

5.1 Analyzing the Commit Timeline

We start our analysis by analyzing the submission timeline of the different groups. GitHub enables the tracking of individual commit history, including name of the user who made the commit, and the date of the commit. We manually tracked the students' commits throughout the semester and the data were indexed in an excel sheet. Extracting this information enabled us to measure the commit frequency by day for each individual assignment. Our findings are presented in Figure 5.2. The figure plots the number of commits made by groups for each day from the day the assignment is assigned to the day it is due. As the results show, for each assignment, the majority of commits happened toward the deadline. An exception of this pattern was the final code assignment, where the maximum number

[1]The grade scale was hidden in compliance with the Privacy Act of 1974

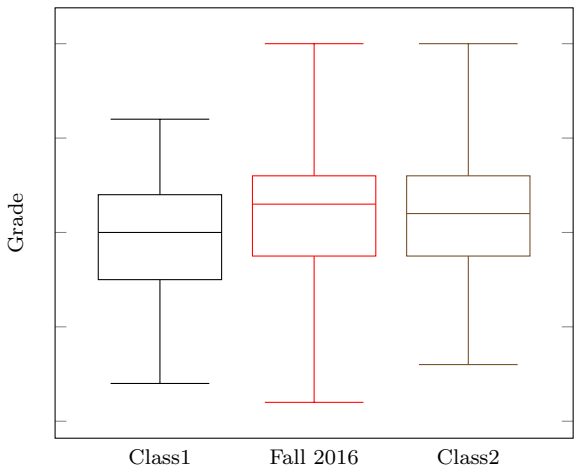
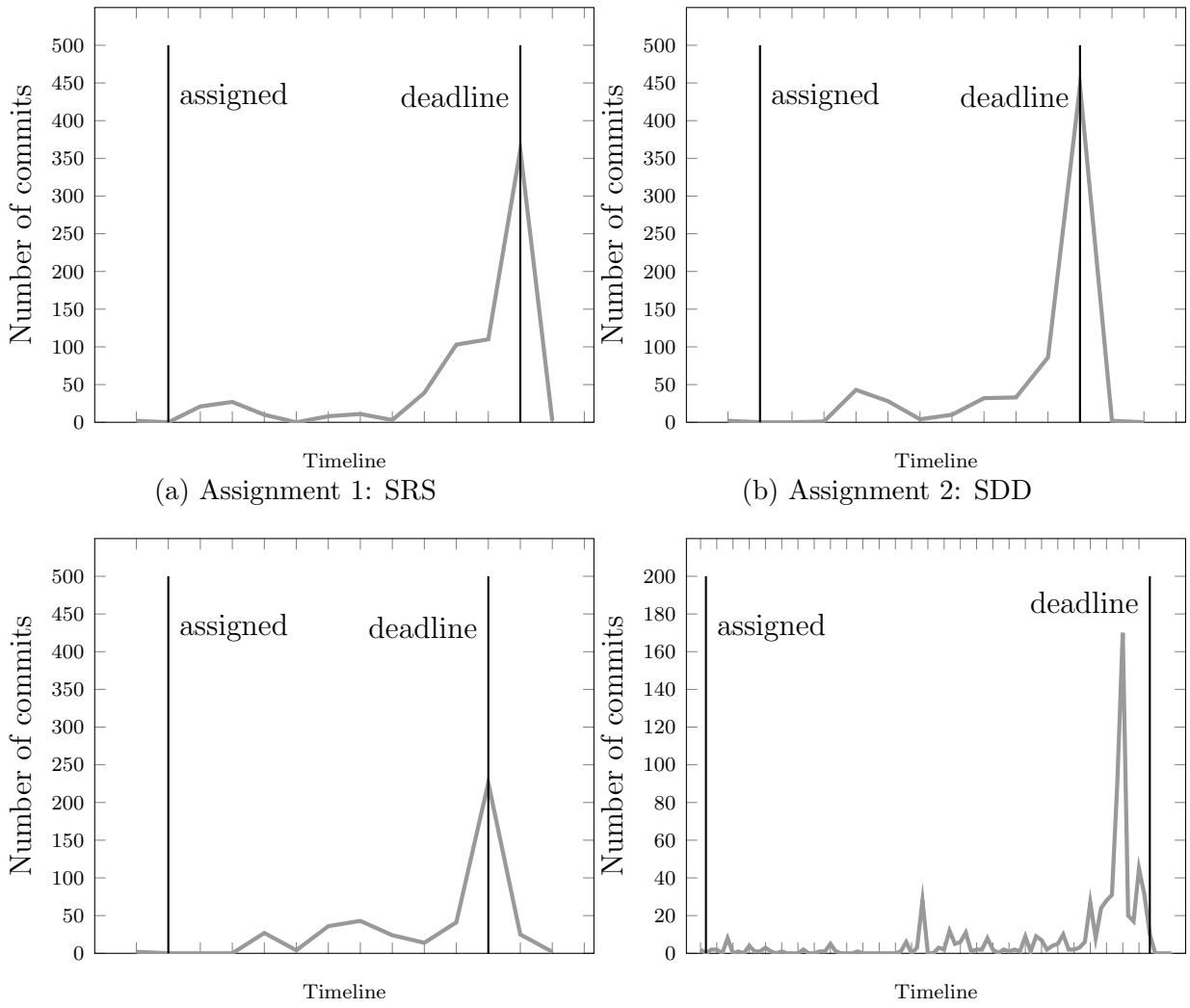


Figure 5.1: Comparing the grades (quality of submission) from the Fall 2016 to two other sections of CSC 4330 where GitHub was not used.

of commits happened five days prior to deadline. These results indicate the presence of academic procrastination. Procrastination is a psychological concept familiar to every person. It results in putting off a task until the very last moment [38, 39]. This behavior is very well-know among undergraduate students. In fact, academic research has reveled that the overwhelming majority of college students are prone to procrastination [40, 39]. In our analysis of assignments' commit history, 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 [41, 42].

In general, our analysis' results have countered our assumption that GitHub would help to control for procrastination. Specifically, although other self-variables (e.g., lack of motivation to learn, lower self-esteem, lack of interest in assignment [43]) have been reported to be related to procrastination, the failure to self-regulate has been found to be the most predictive of procrastination tendencies [43, 44]. In our study, we assumed that GitHub's transparency would motivate our students to contribute by observing others' contributions in the group as well as other groups in class. Furthermore, using a standard collaboration platform which is used by all team members would minimize the high cognitive load that usually results from students adapting various collaboration media and would eliminate the need to standardize, thus enhance the ability of students to self-regulate and hopefully



(a) Assignment 1: SRS

(b) Assignment 2: SDD

(c) Assignment 3: STD

(d) Project Code

Figure 5.2: Total number of commits per day, for each assignment

start working on the assignment [44, 39]. 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

Figure 5.3 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

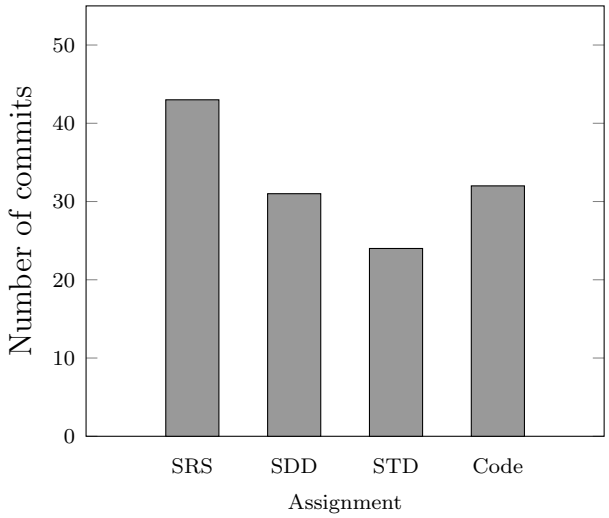


Figure 5.3: The average number of commits for all groups, per assignment

had made fewer and fewer commits. This trend was obvious in the third assignment where the groups made an average of 24 commits in comparison to the first assignment (SRS) where students made an average of 43 commits. In general, as the semester went by, most groups started to self-regulate, meaning that they became more efficient in committing as they figured out the risks of excessive committing. Specifically, in their exit interview, the majority of the students implied that at the beginning they were experimenting with committing and how to resolve merge conflicts, as time went by, they figured out how to regulate the number of commits in such a way that would minimize merge conflicts. Note that the code assignment has a number of commits 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.

5.3 Team Organization

In this section, we explore how different groups adapted GitHub as a part of their team organization (**RQ3**). To carry out our analysis, we tracked and analyzed the commit behavior of each individual student in each group. Our analysis shows that, in general, groups follow three different patterns:

- Equally committing: in 3 out of the 18 groups in our case study, every student in the

team was committing to each assignment throughout the semester. Figure 5.4-a shows a sample commit timeline for one of these groups. The timeline shows that almost all team members were equally active for all assignments. These groups consisted mainly of students at the same technical level. Our analysis of teams' performance shows that these groups tend to do better than other groups as everybody is involved in all aspects of the projects.

- Experience-based committing: in 13 out of the 18 groups, the commit pattern of individual members followed the internal work assignment of the team. Specifically, such groups split the work along different lines: some groups 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). Figure 5.4-b shows a sample commit timeline for one of these groups. The timeline shows how 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 groups reported that they resorted to this structure to work around the variant levels of experiences in the group. According to a group's Project Management Document: *"This style choice allowed for team members to divide project tasks into their own personal area of expertise without the possibility of managerial conflict"*.
- A designated configuration management engineer: in 2 out of the 18 groups, a team member was assigned the role of managing GitHub related tasks. Members of these groups 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 groups indicated that they resorted to this approach to minimize merge conflicts. Figure 5.4-c shows a sample commit timeline for this type of groups. The timeline shows that only one team member was actively committing. Further investigation revealed that these groups tend to follow a chief programmer structure. Under this structure, only one person is responsible for

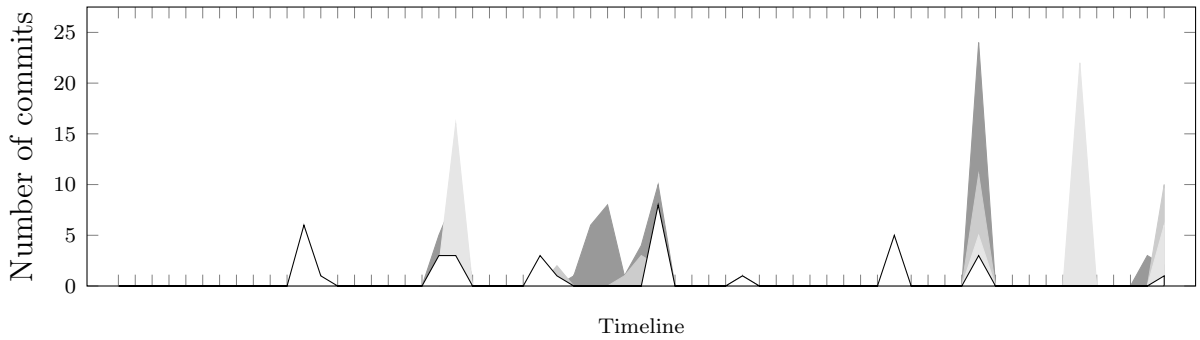
leading the entire project, while others tend to do less work. Our analysis of group performance shows that these groups tend to underachieve in comparison to other, more balanced groups.

A suggestion to balance out the commit effort in unbalanced groups is to consider individual students' GitHub activity as a factor in the student grade. In other words, instead of using a one-grade-fits-all grading scheme, each student is assigned a grade based on his/her GitHub activity (number, size, or quality of commit). However, this strategy might lead to excessive committing, where students will commit just to improve their grade. On the logistic side, objectively analyzing the quality of commits (exact contributions) of different team members can be a tedious job, especially in large class settings.

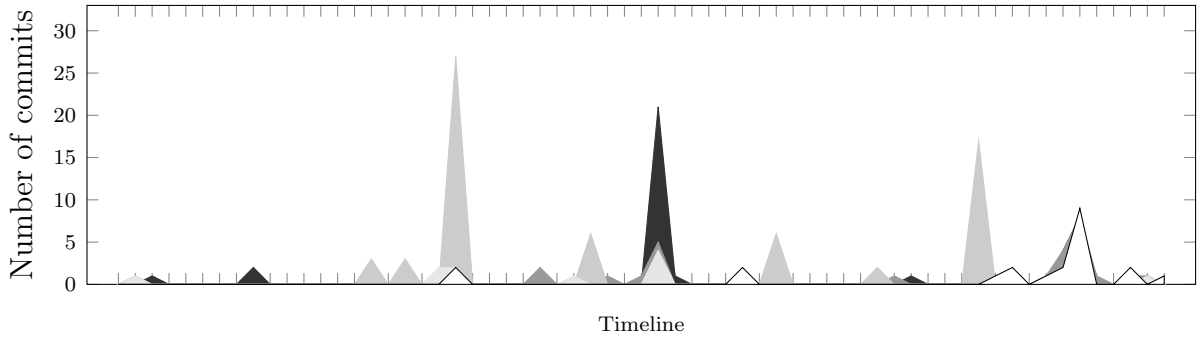
5.4 Number of Commits vs. Performance

Under this part of our analysis, we explore the relation between the students grades and their GitHub activity (**RQ4**). In CSC-4330, a one-grade-fits-all strategy is typically used to grade team projects [3]. Specifically, all team members in the group 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 individual student contributions to their project. In an ideal world, a one-grade-fits-all strategy should enhance the sense of responsibility among students toward their groups as students are aware that their behavior as individuals might affect the overall grade of the group. On the other hand, this grading strategy might encourage free-riders.

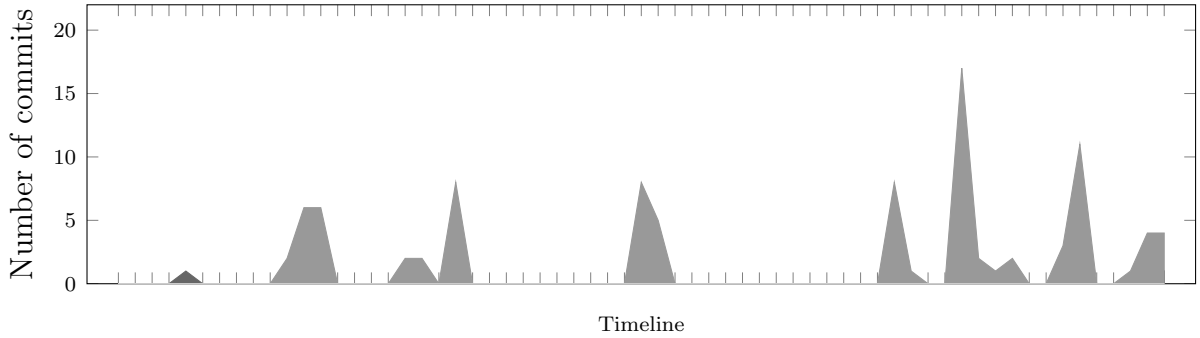
During the semester, all assignments were graded by one of the T.A's and the class instructor according to a pre-defined rubric that had been used before by the same instructor. This rubric does not take into account the students activity on GitHub. The correlation between the number of commits and the group grade is shown in Figure 5.5. The actual grades are omitted from the diagrams to protect students' privacy. In general, the results show that, even though the relationships are slightly positive, no significant correlation is



(a) Equally committing members



(b) Experience-based committing



(c) A designated GitHub engineer

Figure 5.4: Sample commit timelines for 3 different team organization styles. x-axes is time and y-axes is the number of commits

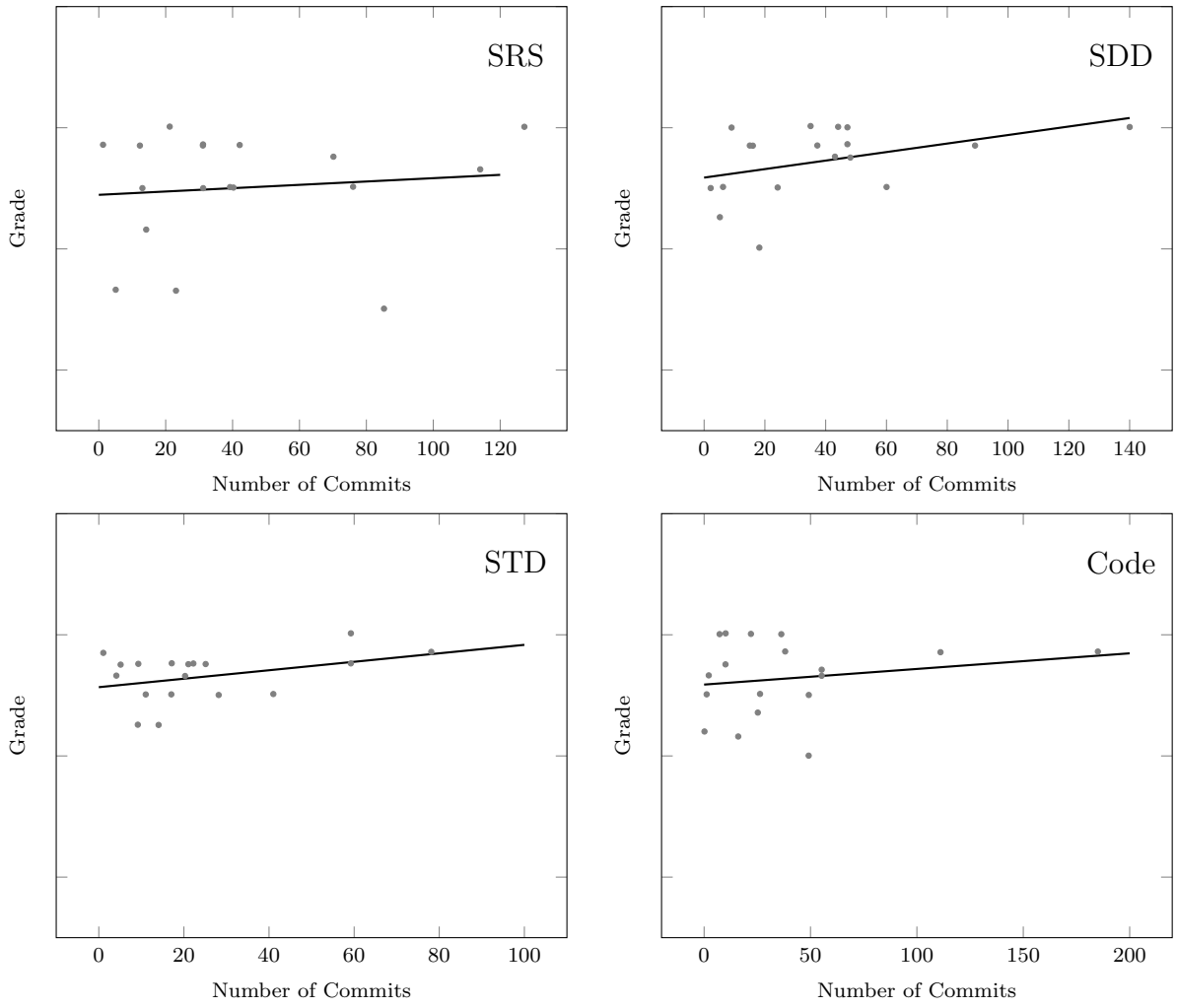


Figure 5.5: Relationship between the total number of commits a team made and the grade received, per assignment

detected between the grade and the number of commits in any of the assignments. In the following section, we discuss these results in greater detail.

Chapter 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*. These platforms have changed the way software being produced and consumed, allowing more and more developers worldwide to join this movement and share their experiences and knowledge in a nontraditional way. In fact, 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 [45].

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 posses [46]. Unfortunately, it is uncommon for CS departments to offer a full class on Configuration Management at the undergraduate level, and sufficiently covering the subject as a 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 CSC-4330. These observations expose an urgent need for research dedicated to identify 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 with references to our main research questions.

- Enforcing GitHub as a collaboration platform in SE team projects can be a very effective way for students to learn about online version control platforms. Given the rapid growth of the OSS movement, these skills are expected to boost students chances of getting a career in CS (**RQ1**).
- 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 (**RQ1**).

- 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 (**RQ2**).
- Different groups adapt GitHub to their workflow differently. In technically balanced groups (all team members are at the same technical level), each member in the group committed to all assignments. These groups, in general, tend to do the best in class. The majority of groups followed a commit pattern based on the different expertise within the team. For example, team members who are good at coding commit to the code assignment, while other group members commit to other documentation assignments. In a few groups, only one team member was assigned to commit for the group. In general, these groups tend to be unbalanced technically and tend to underachieve in comparison to other groups in class (**RQ2**).
- There is no relation between students' grades (quality of submissions) and the number of commits (level of activity). This finding suggests that teachers should be careful before using students' GitHub effort (number of commits, pull requests, and comments) as a basis for grading. On the logistic side, tracking the commit history of individual students can be a tedious job, especially in large classroom settings. Overall, we conclude that GitHub cannot be used as a reliable proxy for tracking or assessing individual student effort (**RQ4**).
- There seem to be a consensus among the students that conventional text editing tools, such as Google Docs or MS Word, can be a more convenient platform for collaborating on documentation assignments (SRS, STD, and SDD). In fact, 9 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 (**RQ2**).

- Some of 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 focusing on these problems early in the semester to speed up the learning process (**RQ2**).
- 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 phone application was the most popular platform. In general, the students reported that they did not feel comfortable communicating publicaly, or even for their internal project discussions to be seen by the class T.A or the instructor. The students also reported that the features that the app provides (e.g., sharing images, videos and other media contents and getting notifications) make 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 overkill for communication (**RQ3**).

Chapter 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 [27]. 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 form 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

Threats to internal validity are influences that can affect the independent variable with respect to causality [27]. Our case study was conducted in a classroom with 84 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 where students would alter their responses just to please the instructor and the TA. For example, claiming to know more programming languages or that using GitHub has positively impacted their learning experience. To mitigate such threats, we enforced anonymity in both of our surveys and the primary class instructor was not present at the time of the surveys. The students were also assured that participating in the survey and their answers did not have an impact on their grades.

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

7.2 External Validity

Threats to external validity are conditions that limit the ability to generalize the results of the experiment [27]. 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 forcing all teams to work on the same project, might lead to a completely different commit behavior. Nonetheless, since our study was specifically aimed at computer science students, and the fact that our study size was reasonable (84 subjects) and our population was diverse, generalization over the whole population of computer science students is still possible. However, we 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 [27]. To measure GitHub experience, we relied on students' assessment of their own skills. It is difficult to objectively evaluate personal knowledge, especially when students do not have enough experience to be able to position themselves among their peers. However, the absolute values are not important here. The shift in perceived experience is what was really important to us.

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 that has been used for 4 years by the same instructor. Furthermore, assignments were graded without taking GitHub activity into account. In fact, to prevent any experimenter bias (the researcher would assign groups who showed more GitHub activity higher grades to support their hypothesis), the researcher who was responsible for collecting GitHub data did not participate in the grading process, also, and groups' GitHub activity was hidden from the graders until the end of the semester.

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 [27]. In our study we compared the student experience in GitHub before taking the class and after it using an independent t-test. Although one of the assumptions of an independent t-test is violated, specifically the independence of observations, the independent t-test was the most appropriate statistical test for our data. One may argue that dependent t-test would be a better choice. However, due to anonymity of the surveys, establishing pairs and pre-survey and post-survey was not possible, therefore, independent t-test was used as a viable alternative.

Chapter 8
Conclusions and Future Work

This paper reports 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 the case study is to explore the way students utilize such a platform to manage their projects and assignments. The research method consisted of a pre-study and a post-study surveys, an exit interview, and a qualitative analysis of students' commit behavior. The results showed that enforcing GitHub in SE team projects can be a very effective way for students to learn about online version control platforms without impacting the quality of their work. However, the results also showed that GitHub cannot be used as a basis for individual effort assessment or to control for problems such as academic procrastination. 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 effort on developing effective educational strategies for integrating an essential software production paradigm in existing CS curricula. To achieve these long term goals, our future work will include conducting similar case studies on using GitHub and other platforms (e.g., SourceForge). Such knowledge will be later used to derive a formal theory that can be used to explain the different factors that control student collaboration in SE class projects, study phenomena such as academic procrastination and free riders, and aid teachers in tasks such as individual student effort assessment.

References

[1] 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.

[2] D. Coppit and J. Haddox-Schatz, "Large team projects in software engineering courses," in *ACM SIGCSE Bulletin*, vol. 37, no. 1, 2005, pp. 137–141.

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

[4] D. Coppit, "Implementing large projects in software engineering courses," *Computer Science Education*, vol. 16, no. 1, pp. 53–73, 2006.

[5] A. Q. Gates, N. Delgado, and O. Mondragón, "A structured approach for managing a practical software engineering course," in *Annual Frontiers in Education Conference*, vol. 1, 2000, pp. T1C–21.

[6] C. Liu, "Using issue tracking tools to facilitate student learning of communication skills in software engineering courses," in *Conference on Software Engineering Education & Training*, 2005, pp. 61–68.

[7] P. E. King and R. R. Behnke, "Problems associated with evaluating student performance in groups," *College Teaching*, vol. 53, no. 2, pp. 57–61, 2005.

[8] O. Seppälä, T. Auvinen, V. Karavirta, A. Vihavainen, and P. Ihantola, "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*. ACM, 2016, pp. 432–435.

[9] A. Goold, N. Augar, and J. Farmer, "Learning in virtual teams: Exploring the student experience," *Journal of Information Technology Education*, vol. 5, pp. 477–490, 2006.

[10] J. Chao, "Student project collaboration using wikis," in *Conference on Software Engineering Education & Training*, 2007, pp. 255–261.

[11] R. Hansen, "Benefits and problems with student teams: suggestions for improving team projects," *Journal of Education for Business*, vol. 82, no. 1, pp. 11–19, 2006.

[12] C. Johansson, "Communicating, measuring and preserving knowledge in software development," Ph.D. dissertation, Ronneby, Sweden, 2000.

[13] C. Colbeck, S. Campbell, and S. Bjorklund, "Grouping in the dark: What college students learn from group projects," *The Journal of Higher Education*, vol. 71, no. 1, pp. 60–83, 2000.

[14] K. O'Reilly, "The management draft: An innovative approach for improving the performance of student teams," in *Marketing Management Association Educators' Conference*, 2015.

[15] A. B. Hollar, "Cowboy: An agile programming methodology for a solo programmer," 2006.

[16] T. Curtis, "So you wanna be a cowboy," *IEEE Software*, vol. 18, no. 2, pp. 112–111, 2001.

[17] R. Slavin, "Cooperative learning," *Review of Educational Research*, vol. 2, no. 50, pp. 315–342, 1980.

[18] D. Williams, J. Beard, and J. Rymer, "Team projects: Achieving their full potential," *Journal of Marketing Education*, vol. 13, no. 2, pp. 45–53, 1991.

[19] J. Buchta, M. Petrenko, D. Poshyvanyk, and V. Rajlich, "Teaching evolution of open-source projects in software engineering courses," in *IEEE International Conference on Software Maintenance*, 2006, pp. 136–144.

[20] N. Clark, "Evaluating student teams developing unique industry projects," in *Australasian conference on Computing education*, 2005, pp. 21–30.

[21] N. Clark, P. Davies, and R. Skeers, "Self and peer assessment in software engineering projects," in *Australasian conference on Computing education*, 2005, pp. 91–100.

[22] R. A. Layton and M. W. Ohland, "Peer ratings revisited: Focus on teamwork, not ability," in *ASEE Annual Conference & Exposition*, 2001.

[23] D. Wilkins and P. Lawhead, "Evaluating individuals in team projects," in *SIGCSE Technical Symposium on Computer Science*, 2000, pp. 172–175.

[24] A. Zagalsky, J. Feliciano, Margaret-AnneStorey, Y. Zhao, and W. Wang, "The emergence of github as a collaborative platform for education," in *ACM Conference on Computer Supported Cooperative Work & Social Computing*, 2015, pp. 1906–1917.

[25] J. Feliciano, M.-A. Storey, and A. Zagalsky, "Student experiences using github in software engineering courses: a case study," in *International Conference on Software Engineering Companion*, 2016, pp. 422–431.

[26] C.-Z. Kertész, "Using github in the classroom-a collaborative learning experience," in *International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2015, pp. 381–386.

[27] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslèn, *Experimentation in Software Engineering*. Springer, 2012.

[28] L. Haaranen and T. Lehtinen, "Teaching git on the side: Version control system as a course platform," in *ACM Conference on Innovation and Technology in Computer Science Education*, 2015, pp. 87–92.

[29] A. Gokhale, "Collaborative learning enhances critical thinking," *Journal of Technology Education*, vol. 7, no. 1, pp. 22–30, 1995.

[30] J. Kelleher, "Employing git in the classroom," in *World Congress on Computer Applications and Information Systems*, 2014, pp. 1–4.

[31] M. Ciolkowski, O. Laitenberger, S. Vegas, and S. Biffl, "Practical experiences in the design and conduct of surveys in empirical software engineering," *Empirical methods and studies in software engineering*, pp. 104–128, 2003.

[32] T. Punter, M. Ciolkowski, B. Freimut, and I. John, "Conducting on-line surveys in software engineering," in *International Symposium on Empirical Software Engineering*, 2003, pp. 80–88.

[33] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause, "An empirical study of practitioners' perspectives on green software engineering," in *International Conference on Software Engineering*, 2016, pp. 237–248.

[34] D. Lo, N. Nagappan, and T. Zimmermann, "How practitioners perceive the relevance of software engineering research," in *Joint Meeting on Foundations of Software Engineering*, 2015, pp. 415–425.

[35] J. C. Carver, L. Jaccheri, S. Morasca, and F. Shull, "A checklist for integrating student empirical studies with research and teaching goals," *Empirical Software Engineering*, vol. 15, no. 1, pp. 35–59, 2010.

[36] A. M. Reisbig, M. H. Jr, M. B. White, and B. R. Rush, "Improving response rates: introducing an anonymous longitudinal survey research protocol for veterinary medical students," *Journal of Veterinary Medical Education*, vol. 34, no. 2, pp. 194–201, 2007.

[37] J. Carver, L. Jaccheri, S. Morasca, and F. Shull, "Issues in using students in empirical studies in software engineering education," in *Software Metrics Symposium*, 2004, pp. 239–249.

[38] G. A. Akerlof, "Procrastination and obedience," *The American Economic Review*, vol. 81, no. 2, pp. 1–19, 1991.

[39] P. Steel, "The nature of procrastination: a meta-analytic and theoretical review of quintessential self-regulatory failure," *Psychological bulletin*, vol. 133, no. 1, p. 65, 2007.

[40] W. J. Knaus, "Overcoming procrastination," *Rational Living*, 1973.

[41] E. D. Rothblum, L. J. Solomon, and J. Murakami, "Affective, cognitive, and behavioral differences between high and low procrastinators," *Journal of Counseling Psychology*, vol. 33, no. 4, p. 387, 1986.

[42] A. J. Howell, D. C. Watson, R. A. Powell, and K. Buro, "Academic procrastination: The pattern and correlates of behavioural postponement," *Personality and Individual Differences*, vol. 40, no. 8, pp. 1519–1530, 2006.

[43] J. Ferrari, "Procrastination as self-regulation failure of performance: Effects of cognitive load, selfawareness, and time limits on working best under pressure," *European Journal of Personality*, vol. 15, no. 5, pp. 391–406, 2001.

[44] R. M. Klassen, L. L. Krawchuk, and S. Rajani, "Academic procrastination of undergraduates: Low self-efficacy to self-regulate predicts higher levels of procrastination," *Contemporary Educational Psychology*, vol. 33, no. 4, pp. 915–931, 2008.

[45] J. Dinkelacker, P. Garg, R. Miller, and D. Nelson, "Progressive open source," in *International Conference on Software Engineering*, 2002, pp. 177–184.

[46] I. Steinmacher, M. Silva, M. Gerosa, and D. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information and Software Technology*, vol. 59, pp. 67–85, 2015.

Vita

Miroslav Tushev, born in Ryazan, Russia, received his Specialist degree from The Russian Academy of National Economy and Public Administration (RANEPA) in 2013. As his interest in computer science grew, he joined the Division of Computer Science and Engineering at Louisiana State University in the fall semester of 2015 to study Software Engineering. While advancing towards the Master's degree, he submitted several papers to multiple prestigious venues, such as IEEE Transactions on Software Engineering, the Journal of Information and Software Technology as well as ACM Transactions on Management Information Systems. Upon the completion of his Masters degree, Mr. Tushev will be continuing his work at the same department to pursue a PhD degree in Software Engineering.