UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Stepan Bolotnikov

Development of a Course on Collaboration Tools in Software Engineering

Master's Thesis (30 ECTS)

Supervisor: Marlon Dumas, PhD

Tartu 2018

Development of a Course on Collaboration Tools in Software Engineering

Abstract:

This thesis describes the creation and delivery of a new course in the Institute of Computer Science of the University of Tartu, titled Collaboration Tools in Software Engineering. The course aims to familiarise bachelor's students with the concepts of Version Control Systems (VCS), Git distributed VCS, issue tracking systems and related collaboration tools, and Continuous Integration (CI). Learning objectives and didactic considerations that shaped the design of the course are explained. Materials that were created for the course and topics that were covered are explained in greater detail. A set of evaluation criteria for the course are presented, the results of the evaluation are analysed and improvements for a potential future iteration of the course are proposed.

Keywords:

Version Control System, Git, issue tracking, GitHub, Continuous Integration, teaching

CERCS: P170 Computer science, numerical analysis, systems, control

Kursus "Koostöövahendid tarkvaraarenduses"

Lühikokkuvõte:

Selles lõputöös on kirjeldatud uue kursuse, Koostöövahendid tarkvaraarenduses, loomise ja õpetamise protsessi Tartu Ülikooli Arvutiteaduse Instituudi jaoks. Kursuse eesmärgiks on tutvustada bakalaureuseõppe tudengitele versioonihaldustarkvara (*Version Control System*, VCS), Git hajutatud versioonihaldustarkvara, ülesannete haldamist (*issue tracking*) ja pidevat integratsiooni (*Continuous Integration*, CI). Seletatakse kursuse õpiväljundeid ja muid kursuse vormingut põhjustanud tegureid ning kirjeldatakse kursuse jaoks loodud materjale, nagu loenguslaidid, videoloengud ja praktikumijuhendid, ning kursuse jooksul käsitletud teemasid. Lõpus on välja toodud kursuse hindamiseks loodud ja kasutatud materjalid, analüüsitud nende tulemeid ja tehtud ettepanekud kursuse võimalike tulevaste toimumiste jaoks.

Võtmesõnad:

versioonihaldustarkvara, Git, GitHub, pidev integratsioon, Continuous Integration, õpetamine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Software Engineering is a wide field that incorporates many scientific and technical fields that deal with the systematic application of engineering to the development of computer software. Software development, in turn, consists of various disciplines and skills, including documenting, testing and programming.

Modern software development projects often involve groups of professionals collaborating on the same set of requirements, documentation, and source code. Thus, tools and formal practices facilitating collaboration must be put in place [Mar09]. Knowledge of the various types of tools and common best practices is equally important when working with large enterprises and academic projects, including group projects that are parts of several mandatory courses for Computer Science, Software Engineering, and other curricula in the University of Tartu.

Important collaboration tools in Software Engineering include Version Control Systems (VCS) and issue tracking tools. In fact, the usage of both is recommended and sometimes even required in several courses involving the development of software projects in groups. However, these courses focus on other aspects of software development and only touch upon the topic of collaboration tools briefly and courses specialising in such tools do not exist in the University of Tartu. This results in students having insufficient knowledge of the working principles, usage, and best practices of the required collaboration tools, which leads to conflicts, confusion, and lowered efficiency in collaboration and intra-team communication as well as decreased quality of the resulting software.

In order to improve the situation, it was decided to create a new course, targeting second-year bachelor's students of the Computer Science curriculum and aiming to present them with an overview of VCS, familiarise them with the usage and best practices of the Git distributed VCS [SC], issue tracking systems, and Continuous Integration (CI). The resulting course was named Collaboration Tools in Software Engineering, assigned the course code LTAT.05.009 in the Tartu University Study Information System, and designed with the European Credit Transfer and Accumulation System (ECTS) score of 3 ECTS in mind [Col]. The course was taught during the spring semester of 2017/18 academic year and was attended by 24 students.

This thesis provides an overview of the work that went into the development and delivery of the course as well as the evaluation criteria and final assessment of the quality of the materials and the course itself.

The rest of this thesis is structured as follows: Section 2 describes the process of course design. The learning objectives and requirements set for the course are discussed. Courses related to the topic, both existing courses in the various curricula in the University of Tartu as well as available open online classes and tutorials are considered. The teaching practices, the structure of the lessons, course schedule and grading criteria of the newly created course are described. Section 3 describes the various types of materials that were

created and the topics that were introduced in the course. Finally, Section 4 presents the criteria for evaluating whether the materials described in Section 3 fill the requirements set in Section 2. The results of the evaluation are analysed and improvements to the course are proposed.

2 Course design

This section describes the process of the development of the course, its goals and structure. Firstly, in Section 2.1, the learning objectives set for the course are described and the motivation behind them is explained. In Section 2.2, related work, including other courses existing in the University of Tartu and online open courses and tutorials are considered and compared against the objectives and goals set for the new course. The schedule of the course and the division of the course into separate topics are explained in greater detail in Section 2.3, giving insight into the delivery process of the course materials. Next, in Section 2.4, didactic considerations that influenced the development of the course and structure of the sessions are explained. Lastly, the grading scheme of the course is presented and explained in Section 2.5.

2.1 Learning objectives

The course created as part of this thesis is meant to teach the participants the working principles, usage, and best practices of Version Control Systems (VCS) and related tools that help software development teams shape the structure of the various information related to projects and effectively manage collaboration between team members.

Collaboration, productivity and project management techniques and technologies are all wide fields that could not be covered in full detail during one course; a narrowing of topics was needed. For this purpose, it was decided that the course would focus on the following topics:

History and motivations behind VCS Before beginning hands-on practical teaching of the usage of a modern VCS, it was decided that the participants should be given insight into the idea of VCS: why such systems are necessary, how they have emerged as a mandatory part of the software development process and how they have developed over time.

Working principles and usage of Git distributed VCS Over time, many VCS have been developed. For the purpose of giving the participants of the course the practical skills and knowledge that would assist them with collaboration in modern software development teams, it was decided to cover a popular modern VCS in greater detail. A state of the industry survey by RhodeCode Inc. [Rho] shows that Git is by far the most popular VCS, so Git was chosen as the main technology to be covered in the course.

Project management and collaboration tools in GitHub Just as with VCS, a great number of software projects focusing on collaboration and project management have been created. It was decided to use a platform that allowed hosting of source code

repositories as well as an integrated suite of collaboration and project management tools, including issue tracking and Continuous Integration (CI) systems. The choice of Git as the VCS to be introduced further narrowed the selection of platform. A report by flow.ci [flo16] introduces and compares several repository management services in terms of popularity, features, license, and pricing. Among them, GitHub [Gitd] is by far the most popular for hosting open-source projects. It also provides the opportunity to host private repositories and has a sufficient suite of collaboration features. Moreover, GitHub allows the students of academic institutions access to the features otherwise reserved for paying customers for free. Because of these reasons, GitHub was chosen as the repository hosting and collaboration platform for the course.

A set of learning objectives was formulated based on the topics introduced above. The course was designed in a way that a participant, upon successful completion of the course, should be able to:

- Explain the purposes and benefits of VCS in software development
- Set up Git and create Git repositories, demonstrate knowledge of branching and common branching models
- Efficiently use Git for source code version management in a team
- Solve common issues with Git, such as merge conflicts
- Understand the principles of issue tracking and how it is used in conjunction with version control
- Be familiar with issue tracking, collaboration and project management tools provided by GitHub
- Have basic knowledge of the principles of CI

As the course focuses on collaboration tools and does not touch upon programming and other activities associated with software development, no specific background knowledge is required from the participants - it is just assumed that the students have basic computer usage skills.

The course's target demographic is second-year students of the bachelor's curriculum in Computer Science at the University of Tartu. By this moment in their studies, the students should have accumulated basic knowledge of software development and encountered some problems that arise when working in groups, but it is likely that they have not yet participated in one of the large mandatory or elective courses that involve the development of a software project in groups, for which the expertise acquired in this course can be very helpful.

2.2 Related work

Because VCS and related collaboration tools are a mandatory part of modern software development process, a great number of learning materials exist that reference the topic in one way or another. Here, such materials are presented. First, in Section 2.2.1, existing courses in the University of Tartu are presented, showing that the existing courses lack in information on the subject. Then, in Section 2.2.2, online learning materials, such as Massive Open Online Courses (MOOC) are considered and compared against the goals and learning objectives described above.

2.2.1 Existing courses in the University of Tartu

While no other course focusing specifically on VCS and other collaboration tools exists in the University of Tartu, a number of courses reference the subject in one way or another; recommending, or sometimes even requiring the use of VCS. Below, some of those courses are named and their connection to the topic is explained.

Web Application Development (code LTAT.05.004) [Web] is a course meant for bachelor's students of the Computer Science curriculum. It is part of the Software Engineering elective module that is recommended for students who wish to continue their studies in Computer Science master's curriculum or work as a software developer after graduation. The participants of the course are divided into teams of three people and are tasked with the creation of a web application. It is expected that the teams use Git for source code version management, but there is no separate guidance given on how to use Git. The students are briefly introduced to some basic Git commands and are provided links to online resources. The course is taught in Estonian.

Software Project (code LTAT.05.005) [Sofb] is a course meant for bachelor's students of the Computer Science curriculum and is also part of the Software Engineering elective module. During this course, students collaborate to create a software project. Usage of VCS, wiki, issue tracker and continuous integration is mandatory, but is not taught in this course - students are expected to find the information on these topics by themselves or ask for help during coaching sessions. The course is taught in English.

Mobile Application Development (code MTAT.03.262) [Mob] is a course meant for bachelor's students of the Computer Science curriculum. It is part of the elective courses module, in which students can pick courses that interest them from a given list. The course includes a number of individual homework and home assignments involving the development of an application for the Android smartphone operating system and some of them require the usage of Git for source code version management and application deployment. Assignment guidelines include a short introductory guide with some basic Git commands, but the topic of Git is not expanded upon. Students are assumed to either be familiar with it already or be ready to learn it by themselves. The course is taught in English.

Object-oriented Programming (code LTAT.03.003) [Obj] is a course meant for bachelor's students of the Computer Science curriculum. It is part of the mandatory module Basics of Programming and is generally taught in the second semester of the first year of the curriculum. The course includes individual and group homework, in which students are introduced to the concepts of object-oriented programming in the Java programming language. It is suggested that students should use Git for collaboration and source code version control. Some external materials on Git are provided, but it is considered more of a bonus and is not further elaborated upon. The course is taught in Estonian.

Automata, Languages, and Compilers (code LTAT.03.006) [Aut] is a course meant for bachelor's students of the Computer Science curriculum and is also part of the mandatory module Basics of Programming. A Git repository is used to provide students with exercises and it is suggested that they use their own repositories for keeping track of their progress, but this is not considered as part of the learning objectives of the course and is not further developed upon. The course is taught in Estonian.

Software Engineering (code LTAT.05.003) [Sofa] is a course meant for bachelor's students of the Computer Science curriculum and is part of the mandatory Software Development module. Students are divided into groups of three people and tasked with the modelling, planning, and development of a software system. Usage of a VCS is required and a short introduction, including links to external self-study materials, is provided, but in general, the students are assumed to acquire the knowledge of the topic by themselves. The course is taught in English.

Agile Software Development (code MTAT.03.295) [Agi] is a course meant for master's students of the Software Engineering curriculum. It is part of the mandatory base module. The course introduces the basics of the agile software development methodology. The course includes homework that has to be done in pairs and a project that is developed in groups of four people. Usage of VCS is expected with Git being implied as the preferred system and students are provided with external study materials, but are expected to learn about VCS by themselves. The course is taught in English.

Enterprise System Integration (code MTAT.03.229) [Ent] is a course meant for master's students of the Software Engineering curriculum. It is part of the Enterprise Software speciality module. The course introduces principles and methods of software architecture in an enterprise environment, with an emphasis on the design, management, and integration of enterprise information systems. Git is extensively used throughout the course for source code version management, continuous integration and deployment. A short introduction to the basic Git commands is given, but this topic is not focused upon. The course is taught in English.

In conclusion, there is a big number of courses currently taught at the University of Tartu that briefly touch upon the topics covered in the Collaboration Tools in Software Engineering course. Many courses involve group projects, for which the usage of VCS is recommended or even required, but none of the courses teach the techniques

and best practices of VCS in detail. Students are usually just provided with a very brief introduction into the basic commands or links to external self-study materials. Having dedicated teaching of the subject in the classroom would allow the students to get feedback and answers to confusing parts from the instructors in a more interactive manner and to improve the general knowledge of the subject.

2.2.2 Learning materials available online

The internet contains a large amount of MOOC, tutorials, articles, books, and other learning materials on every imaginable topic, including VCS, issue tracking, and other related collaboration tools. Below, a number of such materials are named and explained. The materials are compared against the set goals and learning objectives decided for the course developed in this thesis.

The online education platform Udacity has a course titled How to Use Git and Github [Uda]. It introduces the participants to the basics of version control, using Git and GitHub as examples. Through three lessons, participants are taught to create new Git repositories, commit data to them, navigate repositories' commit histories, and collaborate on projects in GitHub. The latter part of the course focuses on creation, review, and merging of pull requests, only touching on issue tracking and other collaboration tools provided by GitHub on a superficial level. The course is available online for free at any time.

Another online education platform, OpenClassrooms, has a similar course, titled Manage Your Code with Git and GitHub [Ope]. It is divided into three parts. The first part introduces the participants to the fundamentals of VCS and Git. Installing Git, making commits and navigating commit history is taught. The second part aims to teach participants how to store Git repositories in GitHub. Collaboration - the last part of the course - focuses on using Git and GitHub for collaboration. This part includes information on branching in Git, tracing changes in Git repositories and contributing to open-source projects via pull requests in GitHub. The course does not include any information on issue tracking and the other collaboration tools provided by GitHub. It is available online for free at any time.

Udemy has a course titled Learning Git - A Beginners Git Course From Infinite Skills [Inf]. The course teaches basics of Git and social collaboration via GitHub in 41 video lectures over 3.5 hours. The topics covered include cloning existing repositories with the SourceTree software and Git command line, making commits, navigating the commit history, branching, GitHub accounts, and creating and merging pull requests in GitHub. Some of the topics, such as commit history, are touched upon at a superficial level and some important topics, such as repository creation, are completely absent. Moreover, the course doesn't go into detail with the collaboration tools provided by GitHub, such as issue tracking. The course is available online for a one-time payment of €48.99.

Coursera has a course titled Version Control with Git [Atlb]. Basic Git usage, including creating repositories, committing, navigating and rewriting history, and merging are covered in the course. Git usage through the Git command line and SourceTree software is introduced and collaborating on Git projects is covered, using the source code hosting service BitBucket as an example. The course does not introduce the participants to GitHub and does not focus on other collaboration tools. It is available online for free with the option of a one-time payment of €39 for access to graded quizzes and a certificate of the passing of the course.

Codecademy has a course titled Learn Git [Cod]. Creating repositories, making commits, navigating commit history, branching and Git remotes are covered in the course. Repository hosting platforms, such as GitHub, are not introduced and other collaboration tools are not touched upon. The course is available online for free. Subscribers of the Codecademy Pro service get access to additional quizzes and exercises for the price of €19.99 per month.

The repository hosting service GitLab has an extensive set of learning materials called GitLab University [Gite]. GitLab University contains links to slides, articles, videos, and MOOC that cover a wide array of topics from basic Git usage to collaboration practices and tools in GitLab. Because the material is sourced from different sources, authors, and websites, it greatly varies in style and quality and can not be considered one cohesive learning material. All of the materials are available for free.

GitHub, in turn, has their own set of learning materials called GitHub Guides [Gitc]. These also cover a wide array of topics but are mainly focused on the social and collaborative aspects of the GitHub platform, going especially into detail about issue tracking capabilities. GitHub Guides does not go into very fine detail when it comes to Git and SCM, instead suggesting people to look to the official Git handbook for further information.

A version of the *Pro Git* book by Scott Chacon and Ben Straub [CS14] is available online for free on the official Git website. The book covers the usage of Git in great detail, starting from basic commands all the way through internal tools. It also touches upon GitHub as a social collaboration platform but doesn't focus on the other collaboration tools.

A ZenHub book, *Better Software & Stronger Teams* [BP16], is available online for free. The book covers the topics of collaboration tools and practices as well as project management with GitHub and ZenHub - an agile project management software that deeply integrates into GitHub. The book is focused on the project management aspect and doesn't touch upon the usage of Git itself.

In conclusion, while many materials, both free and commercial, are readily available online, no one material could be found that fulfilled all the learning objectives set above. The topics could be covered by combining several different courses or books, but this would result in great variation in quality, detail and teaching style.

2.3 Schedule

The course was designed with the European Credit Transfer and Accumulation System (ECTS) score of 3 ECTS in mind. In Estonia, one ECTS credit point is considered to be 26 hours of work completed by the student [RVPV09] via in-class lectures and practice sessions, tests, exams, and independent work, including preparation and study for tests and exams. Thus, the total workload expected of a participant of a three ECTS course is 78 hours.

The course was divided into eight in-class sessions happening every second Friday of the semester. Each session was four hours long and consisted of the lecture part and the practice part.

The learning objectives and materials to be covered were divided between the sessions and are explained below. For each session, the week of the semester, the date and the topic of the session are given.

- Week 2 (February 23rd, 2018): Introduction, History of VCS
- Week 4 (March 9th, 2018): Introduction to Git
- Week 6 (March 23rd, 2018): Repository history
- Week 8 (April 6th, 2018): Branching
- Week 10 (April 20th, 2018): Troubleshooting
- Week 12 (May 4th, 2018): Project collaboration on GitHub
- Week 14 (May 18th, 2018): Git hooks and Continuous Integration
- Week 16 (June 1st, 2018): Guest lecture on Mercurial VCS by Toomas Laasik and preparation for the exam

The course concluded with an exam. Two exam dates were decided: June 4th and June 8th. A detailed description of the material covered in each session and of the exam is provided in Section 3.

2.4 Didactic considerations

The course consists of eight in-class sessions, each four hours long. The sessions were divided into two equal parts - lecture and practice. During the lecture part, the instructor introduced the students to the material to be covered that week. According to the needs of the particular topic, the theoretical background, historical information, and specific tools, software systems, and commands were introduced.

During the practical part, the participants were presented with a set of hands-on exercises on the same topic as that week's lecture, which illustrate the real-world applications of the tools and commands introduced. Students were instructed to complete the exercises independently and submit the results for checking by the next session at the latest. The practice session was held in class with the presence of the instructor so that students could receive help, guidance and further clarifications as needed.

All of the lecture materials and practical exercises were prepared and presented by Stepan Bolotnikov with advice and guidance from Marlon Dumas, PhD. The exception to the usual pattern is the last session. During the lecture part of the eighth session, a guest lecture was given by Toomas Laasik. Mr. Laasik introduced the participants to Mercurial [BLP] - a distributed VCS that can be considered an alternative and a competition to Git. Work principle, repository hosting services, basic commands and typical workflow of a software team using Mercurial for source code version control were presented and illustrated with examples. The practice part of the last lecture was reserved for a recap of the course, preparation for the exam, and discussion. The main practical skills that the students should have acquired during the course were recalled, example exam tasks were presented and students were free to ask questions about the topics that were covered in the course.

2.5 Grading scheme

A non-differentiated final assessment scheme was used for the course. This means that instead of getting a letter grade, participants could only be assessed in one of the three categories: pass, fail, or not present.

In order to get a passing grade, the participant had to fulfil two conditions: actively participate in at least six out of eight sessions and get a mark of "satisfactory" or "good" on the exam. Active participation in this context means that it is not sufficient for the student to just be present at the session; they are expected to participate in the practical part of the session and submit their result to the instructor by the next session at the latest.

The exam covered all of the practical topics introduced in the course and consisted of hands-on tasks. It was open-book, but individual. Students were allowed to use course materials and other online resources, but communication and collaboration on the tasks were forbidden and was grounds for immediate assignment of a failing grade. The exam resulted in the grade of "poor", "satisfactory", or "good" and in order to pass the exam, the students had to get either "satisfactory" or "good". "Good" grade was assigned to students who were able to complete all of the tasks in the scenario presented in the exam guidelines with no mistakes. "Satisfactory" was assigned to students who were able to follow the scenario but made non-critical mistakes, such as failing to consistently follow naming conventions or having a messy commit history. "Poor" was assigned to students who were unable to follow the scenario to the end or made big mistakes, such as failing to use Git branches or committing wrong results. A more detailed description of the

exam is provided in Section 3.

A final mark of "not present" was assigned to students who failed to register for an exam or did not show up for the exam session they registered to.

3 Course materials

A number of study materials were created for the course. The materials helped to guide the in-class activities and helped the students review or study independently in case they missed a session. The materials can also be used in the future as a base for creation and delivery of a similar course.

In this section, the materials created for the course are explained in greater detail. Firstly, a comprehensive list of the types of materials created is presented and elaborated upon. Later, each week of the course is described in terms of the particular topic and material to be covered.

3.1 Types of materials created

Several different kinds of materials were created for the course. First of all, a central material repository was needed in order to grant the course participants easy access to all information. For this purpose, a course website was created in the Institute of Computer Science courses environment [Bol]. Institute of Computer Science courses is a web-based platform using the PmWiki [Yot] content management system for creation of websites for the courses being taught in the Institute of Computer Science of the University of Tartu. It is widely used by the majority of courses in the institute and the students are familiar with it.

PmWiki allows the creation of interlinked pages containing richly formatted text, images, multimedia, and other information. For this course, a website was created that contained the organisational information, contact details of the instructor, guidelines of the practice sessions and examinations, and links to lecture slides, lecture videos, course evaluation quizzes, and external resources.

It was also decided that there needs to be a communication platform that could be used by the instructor to relay announcements about the course to students and by the students to submit their practice session results, get feedback on them, ask questions, and hold discussions about the course and its contents with the instructor and their peers. Presently, it is common for courses in the Institute of Computer Science of the University of Tartu to have an instant messaging platform set up for these purposes. For example, the course Automata, Languages, and Compilers uses the Fleep messaging software and courses Software Engineering and Software Testing use the Slack messaging software.

As Slack seemed to be the most common option and thus can be assumed to be the most familiar to students, it was decided to set up a Slack workspace for the course. The information about the Slack workspace was provided to the students via lecture slides and course website. In the majority of the practice sessions, the students were asked to submit the link to the final result to the instructor in Slack. This is also where they received the feedback on their submissions.

3.1.1 Lecture slides

The first half of each session was reserved for a lecture. Depending on the particular topic, the lectures included information about the theoretical background, tools, techniques, and technologies, or historical facts. A set of slides was created for each lecture in order to enforce a structure and flow of information throughout the session and assist the students with revision outside of the classroom.

The slides were created with the Google Slides [Goo] software, with some additional formatting done with Microsoft Powerpoint. Before each lecture, the slides were converted to the Portable Document Format (PDF) and uploaded to the course website, where they could be downloaded and viewed by the students.

3.1.2 Lecture videos

Because the classroom where the lectures were being held was equipped with a webcam and a microphone, it was decided to make a video recording of every lecture and upload it to the University of Tartu Panopto platform [Pan]. Panopto is an online video recording and content management software in use by the University of Tartu for sharing and management of video lectures.

Lecture recordings done with the Panopto software include screen and slide recording in addition to webcam video and voice. This allows for the creation of rich multimedia recordings that let the viewers navigate the videos by slides and chapters, see the slide that the instructor is talking about and see on-screen demonstrations.

The lecture recordings give additional insight into the course delivery procedure after the fact. This gives students additional context and richer experience when reviewing lecture topics or studying the material of a missed lecture independently. It also provides additional context to the lecturer for reviewing their own performance during the lecture and noticing ways to improve in future lectures or in the next iteration of the course. Lastly, the lecture recordings can be used to transform Collaboration Tools in Software Engineering into an online course.

The recording was uploaded to the University of Tartu Panopto service after every lecture and when ready, the link to the video was added to the course website.

3.1.3 Practice session guidelines

The second half of each session was reserved for the practice session. During this part, the students were supposed to individually and independently complete a set of hands-on exercises that were connected to the themes that were introduced in the lecture.

A page on the course website was created for every practice session and a link to the page was added to the list of lessons before the practice session. The page contained a list of exercises, guidelines, and explanations about that week's practice session.

Some practice sessions involved the creation of a Git repository, in which case no additional material was given. However, some practice sessions required the students to work on an existing repository. In the latter case, a link to the repository hosted in GitHub was also provided. If students needed to be able to work on that repository directly - as opposed to making a copy of the repository and working with the copy - students were granted collaboration access to the repository before the practice session.

Active participation in practice sessions was one of the two main requirements to get a passing grade for the course. Active participation is defined as the students completing the tasks of the practice sessions and submitting them to the instructor for checking. If the students made mistakes in their solutions, they were not marked as absent or asked to re-do the tasks. Instead, they were simply given feedback on their submission through the course's Slack workspace and asked to review the parts of the lecture that they did not fully understand.

3.1.4 Quizzes

For each session, except for the first and the last, a quiz was prepared and the students were asked to complete it. The quizzes consisted of multiple choice questions about the topics covered in that week's lecture and practice session and served two purposes. Firstly, the quizzes were to be used as self-evaluation so that students could check if they understood that week's learning goals and see what parts they would need to review. Secondly, the results of the quizzes were used as feedback and were used in the evaluation of the course to identify what topics students found to be the hardest and delivery of what topics should be improved upon in potential future iterations of the course. The analysis of the feedback to the course is presented in Section 4.

The quizzes were created using the SurveyMonkey [Sur] online survey platform. The free version of SurveyMonkey allows creating unlimited surveys with up to 10 questions, share them with the respondents, collect and effectively analyse answers. A new survey was created for each session and the link to it was added to the practice session guidelines.

3.1.5 Exam

Two exam dates were planned for the course: on the 4th and the 8th of June. Initially, later dates were considered, but an earlier date was settled for. This was due to the fact that there were several third-year bachelor's students attending the course and in order to graduate that semester, they needed to get all marks by the 5th of June at the latest.

The exam was designed to be completed individually over the course of approximately 1.5 hours [Bol18]. It consisted of practical tasks that covered all of the topics introduced in the practical sessions throughout the course.

A Git repository was created for the exam and hosted on GitHub. The repository included some sample code, commits, issues and pull requests. The students were tasked with picking an unassigned issue, fixing it, doing some exercises with the history of the repository and submitting their changes as a pull request. As the last step, they were required to pick one of the pull requests previously created by the instructor, conduct a code review, and decide whether the pull request was to be accepted or rejected. Because the goal of the exam was to demonstrate the students' knowledge of Git, not software development, the issues created for the exam repository did not rely on prior knowledge of any programming language and were mostly simple changes of one row in the existing files.

It was possible to get a mark of "poor", "satisfactory" or "good" for the exam. "Good" was awarded to students who were able to follow the scenario of the exam and did not make any mistakes with the tools and techniques covered by the course. "Satisfactory" was awarded to those who were able to follow the scenario but made minor mistakes, such as not following the naming convention set for branches or failing to make commits at regular times, resulting in messy commit history in their submission. If the student was unable to follow the scenario or made major mistakes that suggested that the student did not acquire the skills and knowledge covered in the course, they were graded with "poor".

A grade of "satisfactory" or "good" on the exam was one of the two main prerequisites to passing the course, with the other one being active participation in at least six out of eight practice sessions. If the student got a "poor" grade for the exam, they received a mark of "failed" for the whole course. If a student did not register to either of the two exam sessions or failed to attend the session they were registered for, they received a mark of "not present" for the whole course.

3.2 Topics

The creation of the materials for the course combined materials from several distinct sources by different authors into one comprehensive set of learning resources that covered all of the topics outlined in the learning objectives of the course.

The working principles and usage techniques of the Git distributed Version Control System (VCS) was mostly sourced from *Pro Git* [CS14] and the official Git documentation [Gita]. Some additional information, especially regarding industry standard best practices, was sourced from articles by companies and individuals involved with the development and maintenance of VCS and VCS repository hosting services.

The information about issue tracking, Continuous Integration (CI), and other collaboration tools was mostly sourced from *Better Software & Stronger Teams* [CS14] and GitHub Guides [Gitc]. The source materials used in a lecture are listed on the last slide of each lecture's slides.

Below, the contents of each week of the course - the themes covered in the lecture and the tasks solved during the practice session - are explained.

3.2.1 Introduction, history of Version Control Systems

The first week of the course was focused on covering two topics: introduction to the course and introduction to VCS. First, the organisation of the course was explained. The students were familiarised with the lecturer and the structure, schedule, learning objectives, and evaluation criteria of the course.

Later, introduction to VCS was given. The need for VCS-like software in modern software development processes was explained. The lecture materials covered the earliest known VCS and illustrated the development of the technology through time, dividing it into three commonly recognised generations: local VCS, central VCS, and distributed VCS [dAS09]. For each generation, the working principles and differences from the previous generations were explained, the most well known VCS of that generation were named and a short example of usage was given. As the last topic of the lecture, Git was introduced. A short history of Git's development was presented, as well as its unique features and its main advantages over other contemporary VCS.

The practice session of that week introduced the students to Git and GitHub. The students were tasked with downloading and installing Git on their computers, registering a GitHub account, creating their first repository, and adding a file to it - this kind of a short introduction is also often given in the other courses where students are expected to use a VCS. This served as an introduction to the platforms and technologies that were used throughout the course. When the practice session was done, students were asked to send the link to their repository to the instructor via the course's Slack workspace in order to be counted as an active participant of the practice session.

The materials for this week were sourced mainly from *Pro Git* [CS14], *Version Control by Example* [Sin11], and an article [San10] by Pablo Santos from Códice Software, a company developing the Plastic SCM distributed VCS. The example of usage of Source Code Control System was taken from *Programming Utilities Guide* [Sun97] and the example of usage of Subversion was taken from the Apache Subversion documentation [Apa].

3.2.2 Introduction to Git

The second week of the course was mainly focused on introducing the basic usage of Git in greater detail. Commands relating to the configuration of Git, creation and cloning of Git repositories, checking changesets in and out of the repository, and synchronising remote repositories were presented and explained. Students had already used some of the commands, such as `git clone`, `git add`, and `git push`, in the previous practice session,

but during this week's lecture they learned in greater detail what those commands are used for and exactly what happens when they are invoked.

The first topic of the lecture was Git configuration. The `git config` command and types, syntax, and location of the files holding Git configuration parameters were explained. As an example of what Git configuration is often used for, the concept of Git aliases was introduced and the creation of aliases was demonstrated.

During the previous practice session, the students used the `git clone` command to download a remote Git repository into their computer. This time, the differences between the `git clone` and `git init` commands and the process behind each was explained. The application of "bare" repositories, common scenarios for creating a repository to be synchronised between two machines, and usage of remote repositories and the `git remote` tool were also explained.

The last part of the lecture was focused on the most common and essential Git commands in detail. The life-cycle of a file in a Git repository and the staging area specifically were explained. The commands for staging and resetting changes, checking changes in and out of the repository, removing files from the repository, and getting newer data to and from remote repositories were introduced. The usage of a `.gitignore` file in Git repositories was also demonstrated.

In the previous practice session, it became apparent that a number of course participants were not sufficiently familiar with the Unix command line. Because the main tool for Git usage in the course is Git Bash, which mimics the working principle and commands of a Unix command line, it was decided to start this week's practice session with a short reminder of the basic Unix command line tools for navigating and manipulating the file system. The `pwd`, `ls`, `cd`, `mv`, `cp`, `rm`, and `rmdir` commands were introduced.

The rest of the practice session consisted of practical exercises that required the students to use all of the techniques and commands that were introduced in the lecture. The students were instructed to create a repository, make changes to the configuration of their local Git installation, set up a `.gitignore` file, make some changes to the files in the repository, and synchronise the changes with the remote repository hosted on GitHub. In order to be counted as an active participant in the session, the students had to send the instructor a link to the GitHub repository at the end of the session.

The materials for this week were largely sourced from *Pro Git* [CS14].

This was the first week where students were asked to complete a quiz after the practice session. The quiz was meant to be used for self-evaluation and as feedback to the instructor. Another objective of the quizzes is to highlight the topics that were difficult for the students, which is information that will be useful for improving the materials in a potential future iteration of the course. The results and analysis of the quizzes is presented in Section 4.

3.2.3 Working with repository history

The third week of the course was focused on the tools for browsing and modifying the commit history of a Git repository. The way commits are held and managed in a Git repository was explained. In addition, the `git log` and `git show` commands for viewing the commit log were introduced and changing existing commits via the `git rebase` command and the `--amend` argument of the `git commit` command was explained.

Firstly, the structure of Git commits was explained. The students were demonstrated what information is contained in a commit and how commits are uniquely identified by their Secure Hash Algorithm 1 (SHA-1) hash digest. The working principle of the SHA-1 algorithm was explained briefly, but the main focus was on the way SHA-1 is used by Git to ensure data integrity, not the cryptographic and security applications of the algorithm, so its implementation was not further elaborated upon.

The `git log` tool was introduced for displaying information about the commit history of a Git repository. Different arguments for filtering commits to be displayed and the format in which the data can be output were demonstrated. Examples of a commit log of a real Git repository were used to illustrate the usage of the command and its arguments. The usage of the `git show` command for viewing information about one commit was explained. Some external tools for browsing the commit history of a repository, including the tools provided by the GitHub web interface, were demonstrated.

The usage of the `git checkout` command for reverting the working tree of the repository to a previously committed state was demonstrated, including the way to restore a file that had previously been deleted.

Lastly, the `git rebase` and the `--amend` argument of the `git commit` command were introduced. Examples were used to show how the `git rebase` command can be used to modify the commit history of a Git repository by changing the content or order of commits and deleting commits. The usage of the `--amend` argument of the `git commit` command for changing the latest commit in a repository was demonstrated. The dangers of changing commits that had already been synchronised with remote repositories were explained.

The previous practice session showed that several students had problems with understanding the differences between a normal repository and a "bare" repository. At the beginning of this week's session, the differences were explained again and demonstrated with examples.

The rest of the practice session of this week consisted of a set of practical tasks that the students were tasked with solving by using the commands and techniques introduced in the lecture. A Git repository with several commits was created and hosted on GitHub. The students were introduced to the practice of forking repositories to copy repositories to their own GitHub account.

The students were then given several questions and tasks that could be solved by using the commands that were introduced in the lecture. The tasks included navigating the

commit history of the repository, restoring a previously deleted file, and modifying history with the `git rebase` command. After solving all the tasks, the students were required to send a link to their repository to the instructor via the course's Slack workspace and answer a self-evaluation quiz that was created for this session.

The materials for this week were largely sourced from *Pro Git* [CS14].

3.2.4 Branching

In this week's lecture, the concept of branches in VCS and the specifics of branches in Git were explained. The commands for manipulating branches were introduced: `git branch` for viewing, creating, and deleting; `git checkout` for switching the active branch; `git merge` for merging two histories together.

The usage of the `git log` command that was introduced in the previous lecture was revisited and expanded upon by introducing new arguments that can be used to view the history of a Git repository that has multiple branches. The concept of Git aliases was also revisited and its application was demonstrated through the creation of a short and easy to remember alias to an otherwise long and elaborate command.

The students were then introduced to the branch workflow and explained what branches are used for. The different kinds and naming conventions of branches that are used in popular Git branching workflows were explained with the example of a successful branching model introduced by Vincent Driessen [Dri10].

Lastly, viewing and switching branches in the GitHub web interface were demonstrated and the students were provided with additional reading materials about alternative Git branching models.

In this week's practice session, the students were tasked with using the tools that were introduced in the lecture for manipulating branches in an existing repository. A Git repository with several commits and branches was created and hosted on GitHub. The students were instructed to fork it, answer some questions about the branches in the repository, create, merge, and delete branches. Upon completion, the students were required to send the link to their Git repository to the instructor via the course's Slack workspace and fill the self-evaluation quiz.

The materials for this week were largely sourced from *Pro Git* [CS14] and A Successful Branching Model [Dri10].

3.2.5 Troubleshooting

The session of the fifth week focused on two main themes. Firstly, the lecture expanded on the topic of branching that was introduced in the previous lecture by explaining what may cause merge conflicts in a Git repository and what steps need to be taken to solve merge conflicts. Secondly, the `git blame` and `git bisect` commands were introduced and their application to debugging code by tracing the origin or changes was explained.

First, the content of the last lecture was briefly revisited by going through a scenario where the history of a Git repository branches out and the branches eventually merge with no conflicts. Then, a modified scenario was presented to demonstrate a merge conflict. The differences between the two and the origins of merge conflicts were explained. The steps that can be taken to abort the merging process, or to begin to solve a merge conflict were then presented. Conflict resolution markers were introduced and the procedure of manually solving a merge conflict was demonstrated. In addition to that, tools for solving merge conflicts that are part of the IntelliJ IDEA Integrated Development Environment (IDE) were demonstrated to show how merge conflicts can be solved by using tools commonly provided by IDEs. IntelliJ IDEA was chosen as an example because it is a popular and fully-featured IDE that is also being recommended and used in many programming-related courses in the Institute of Computer Science of the University of Tartu, so it should be familiar to the students.

Because a Git repository has the information about when, by whom, and how the code was changed, Git was proposed as a helpful tool for debugging. For this purpose, the `git blame` command was introduced for tracing changes made to a file that is being tracked by a Git repository. Arguments of the command that allow limiting the output were explained and usage of the command through the tools integrated into IntelliJ IDEA was shown.

Lastly, the `git bisect` command was introduced for doing a binary search through the history of a Git repository. A scenario that illustrated the application of the command was described.

Instead of having each student making a copy of a repository, this time all students were granted collaborator access to the repository that was created for the practice session. Each student then added a directory and a text file to the repository. The students were tasked with answering some questions about the commit history of the repository and origin of changes in code files present in the repository. The answers to the questions could be found by using the `git blame` command that was introduced in the lecture. The repository contained an image file. The students were asked to use the `git bisect` tool to find the first commit in which the colour of the image had changed. Then, the students were to simulate a scenario that resulted in a merge conflict and resolve that conflict. Lastly, the students were to add their name to a text file. As everyone was working in the same repository and the names were added to the same file, for some students that resulted in a merge conflict that they were then supposed to resolve.

After all mentioned tasks were completed, the students were instructed to push the changes that they made to the remote repository and fill the self-evaluation quiz.

The materials for this week were mostly sourced from *Pro Git* [CS14] and IntelliJ IDEA help website [Jet].

3.2.6 Project collaboration in GitHub

This week's lecture focused on collaboration tools other than Git. The concept of issue tracking was introduced and collaboration and project management tools provided by the GitHub platform were explained.

Firstly, the need for issue tracking software and consistent usage thereof was explained. The common attributes of issue tracking systems were presented and the different kinds of issue tracking solutions were demonstrated and exemplified with specific products. They were compared in terms of flexibility, features, and integrations into code repositories.

The GitHub platform was used as an example of a collaboration and project management platform. The features of the issue tracking system that is part of GitHub's web platform were explained. The application of GitHub's project boards to organising issues was demonstrated. The concept of pull requests was introduced and applied to the collaboration workflow that is being used on GitHub and other modern repository hosting services. Lastly, creating pages in GitHub wiki was demonstrated.

For this week's practice session, a repository with some code was created and hosted on GitHub. A number of issues were created using the GitHub issue tracker of this repository. Students were granted collaborator access to the repository. The students were to pick an issue that had not been assigned to anyone, assign it to themselves, make the changes necessary to fix the issue, and open a new pull request with their changes. They were then to request another student to conduct code review of their work and in turn review someone else's work. At the same time, the students were to reflect the status of their issue in a project board of the repository and document their actions in a new wiki page they had created for themselves. This scenario took the students through the whole suite of collaboration and project management tools offered by the GitHub platform.

Finally, the students were also to fill the self-evaluation quiz.

The materials for this week were mostly sourced from the GitHub website [Gitb], GitHub Guides [Gitc] and *Better Software & Stronger Teams* [BP16].

3.2.7 Git hooks and Continuous Integration

In the seventh lecture of the course, the concept of Continuous Integration (CI) was introduced and Git hooks were explained as a way to automate tasks with Git.

Firstly, the need for automating certain parts of the software development was explained and CI as a service for automating testing and building of applications was introduced. The typical features of a CI service were listed and the different ways of setting up CI were presented. Git provides hooks - a way to launch scripts when certain events in repository occur. Git hooks were proposed as a way of setting up CI.

The different types of events that can trigger Git hooks were presented. For each

hook, the triggering event and common applications were discussed. The technical details of creating a Git hook were explained and an example Git hook was demonstrated. An example setup of a CI system with Git hooks was presented.

Finally, some popular CI tools and the ways of setting up a CI service with a GitHub repository were presented. Travis CI was used as an example.

This week's practice session was mostly about setting up Travis CI with a GitHub repository and creating a Git hook to demonstrate how they work. The students were to create a new repository, add some example code, sign up for a Travis CI account, and set up automated testing for the example code with Travis CI. An example automated test for a script written in the Python language and an example Travis CI configuration file for an application written in Python were given. The Python script had a small error, which the students were meant to find and fix with the help of automated testing with CI. The students were then to add an image to the `readme` file of the repository that displayed the current state of the CI. Finally, the students were tasked with using the same test script to create a `pre-commit` hook in their repository and inspect how it works.

After completing all the tasks, the students were required to send the link to their repository to the instructor via the course's Slack workspace and fill a self-evaluation quiz.

Travis CI was chosen as the CI service for this practice session for several reasons. Firstly, it is delivered as a Software as a Service (SaaS) and doesn't need to be installed and configured separately. Secondly, it is free for all open-source projects and thirdly, it directly and seamlessly integrates with GitHub repositories.

The materials for this week were mostly sourced from *Pro Git* [CS14], the GitHub website [Gitc], and Travis CI website [Tra].

3.2.8 Guest lecture, preparation for exam

For the last week of the course, a guest lecturer - Toomas Laasik (Interactive Fate OÜ [Int]) - prepared and delivered a lecture about Mercurial. Mercurial is another distributed VCS that can be considered an alternative and competition to Git [Tho08]. Mr. Laasik explained the main differences between Git and Mercurial, demonstrated the usage of Mercurial and presented popular collaboration tools that can be used with it, such as repository hosting services and issue tracking tools. The lecture was illustrated with examples of real software projects that used Mercurial for source code management. This topic was chosen to give the students a wider view of the state of the industry regarding the usage of VCS and give an in-depth view into a serious alternative to Git by an industry professional with years of experience working with teams that use Mercurial as primary VCS.

The practice session of the last week was reserved for preparation for the exam. The main techniques and tools that were covered in the course were revisited, example exam exercises were solved and a discussion about the course in general was held, during

which the students were able to give feedback to the instructor and ask questions about parts of the course that seemed vague or confusing or about related topics that were not covered in the course.

4 Results and course evaluation

This section describes the process and materials that were used to evaluate the quality of the course and judge the fit of the materials described in Section 3 to the objectives described in Section 2. Results of the evaluation are analysed and conclusions are drawn, identifying the weak points and suggesting improvements for potential future iterations of the course.

4.1 Course evaluation criteria

Because the deadline for the submission of master's theses was earlier than the end of the semester, formal feedback that is usually collected by the University of Tartu at the end of each semester and exam results were not available at the time of writing of this thesis. Consequently, a set of quizzes and a final questionnaire were created to be used for the purpose of the evaluation of this course. A quiz was created for each week of the course, excluding the first and the last week. The first week was excluded due to the mostly introductory nature of the lecture and the last week was excluded because it consisted of a guest lecture and revision of the course materials in preparation for the exam.

The quizzes were made up of multiple choice questions about the materials covered in the corresponding lecture. In addition to feedback for the course, the quizzes also served as self-evaluation for the students. The full list of quizzes and the questions that they consisted of is presented in Appendix I.

The final questionnaire consisted of questions that were meant to collect the students' opinions and evaluations of the course. The questions were sourced from the Berkeley Center for Teaching and Learning Course Evaluations Questions Bank [Uni]. Questions were selected based on how they fitted the content and structure of the course. An additional open-ended question was added. The questions were either open-ended, used a scale from 1(lowest) to 5(highest), or used a five-level Likert-type scale to collect the students' attitudes towards the course. The questionnaire included just 14 questions, which is less than the formal course feedback form that is used by the University of Tartu, to avoid fatigue effect. The questions are presented in Table 1.

The free version of the SurveyMonkey platform, which was used for the weekly quizzes, was not used to create the questionnaire because of its 10-question limit. Instead, ProProfs [Pro] was used for the final questionnaire. ProProfs offers a similar service and a subscription to the platform was available courtesy of Marlon Dumas, PhD, the supervisor of this thesis.

The questionnaire was filled by the students at the end of the practice session of the seventh week. Because the questionnaire was filled in class, it actually yielded more data than could have otherwise been expected of the formal university feedback, as it is optional and is filled by the students in their free time.

Table 1. The final course feedback questionnaire

Number	Question	Type
1	The instructor explained concepts clearly	Likert-type
2	The instructor was helpful when I had difficulties or questions	Likert-type
3	The instructor presented content in an organized manner	Likert-type
4	The instructor provided clear constructive feedback	Likert-type
5	The instructor encouraged student questions and participation	Likert-type
6	How would you rate the overall effectiveness of the instructor's teaching?	1-5
7	The course was effectively organized	Likert-type
8	The course instructions (including, manuals, handouts, etc.) were clear	Likert-type
9	The course developed my abilities and skills for the subject	Likert-type
10	The course developed my ability to apply theory to practice	Likert-type
11	How satisfied were you with this course?	1-5
12	Please identify what you consider to be the strengths of the course.	Open-ended
13	Please identify area(s) where you think the course could be improved.	Open-ended
14	Please write any other comments or suggestions you have regarding the instructor, the course, the materials, etc	Open-ended

4.2 Analysis of feedback

Firstly, the quizzes are analysed. For each quiz, the overall completion rate and the average score are shown. The questions that received the most incorrect answers are looked at in greater detail. The full set of data that was collected via the quizzes is included with this thesis in Comma-Separated Values (CSV) files `week_2_quiz_responses.csv` through `week_7_quiz_responses.csv`.

Secondly, the feedback questionnaire is analysed. The students' overall satisfaction with the course is inspected, problematic places are identified and suggestions are considered. The full set of responses to the final questionnaire is included with this thesis in the file `feedback_questionnaire_responses.csv`.

4.2.1 Quizzes

The second week covered the basic usage of Git and the quiz consisted of questions about the tools that were introduced during that week's session. The quiz was filled by 13 students and had an average score of 66%. The distribution of score is shown in Figure 1. Only one student got all answers right and no question was answered correctly by everyone.

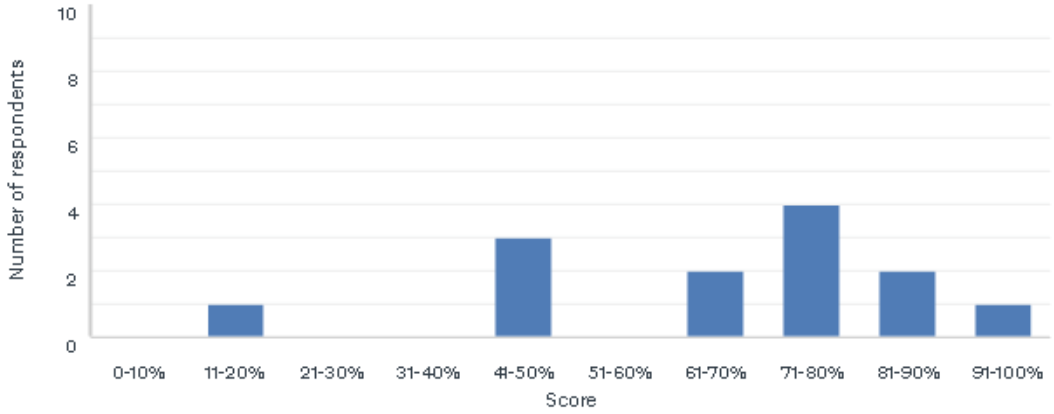

Figure 1. The distribution of scores in the quiz for the second week

The hardest question was question number one: "What is NOT one of the ways to change Git configuration?". The question had five options:

1. Manually edit the `/.gitconfig` file;
2. Manually edit the `.gitconfig` file in the repository;
3. Manually edit the `/etc/gitconfig` file;
4. Use the "`git config`" command;
5. Manually edit the `.git/configfile` in the repository.

Out of them, the second one was correct and was chosen by five people, but options number three, four, and five were also popular choices (two, two, and four people, respectively). This suggests that the topic of different files that hold Git configuration parameters remained confusing to the students. While they also received some wrong answers, the other questions of this quiz had a much higher average score, as seen in Figure 2.

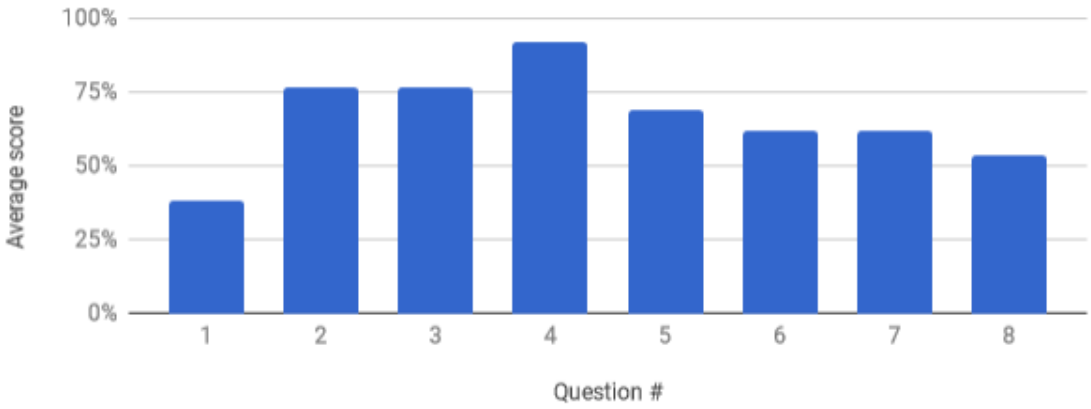

Figure 2. The average score per question of the quiz of the second week

The third week covered the navigation and manipulation of the commit history of a Git repository. The quiz was filled by 16 students and had an average score of 95%. Score distribution was considerably better than in the second week (see Figure 3). Five out of Eight questions were answered correctly by all students and 12 out of 16 students answered all questions correctly.

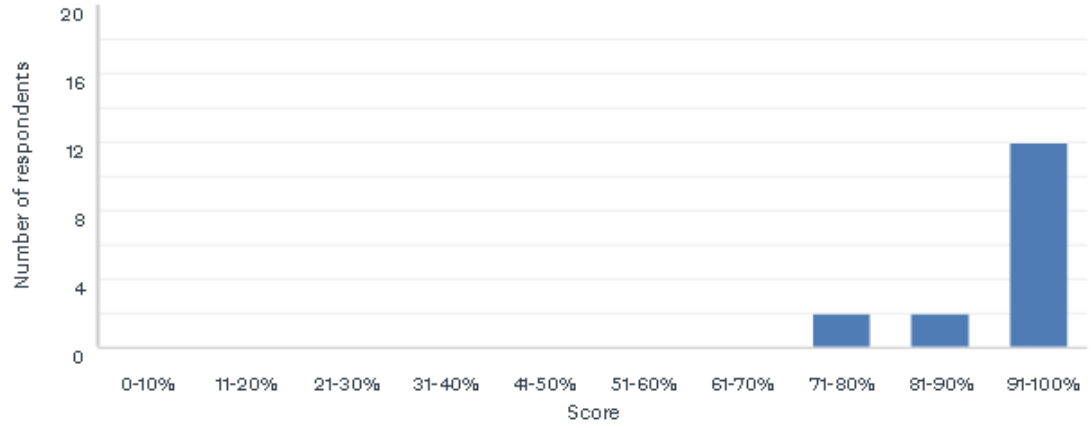

Figure 3. The distribution of scores in the quiz for the third week

The most challenging question, with 3 students having answered it incorrectly, was question four: "What command is used to change the last commit?". The correct answer was "`git commit --amend`", but three students chose "`git commit --rebase`". The `git commit` command doesn't have a `--rebase` argument and the answer was

specifically designed to see if the students had paid attention and remembered the difference between the `--amend` argument and the `git rebase` command, both of which were introduced in the lecture. The feedback of one student suggested that they would have wished to see a longer or more thorough demonstration of the usage of the `git rebase` command. The average score per question can be seen in Figure 4.

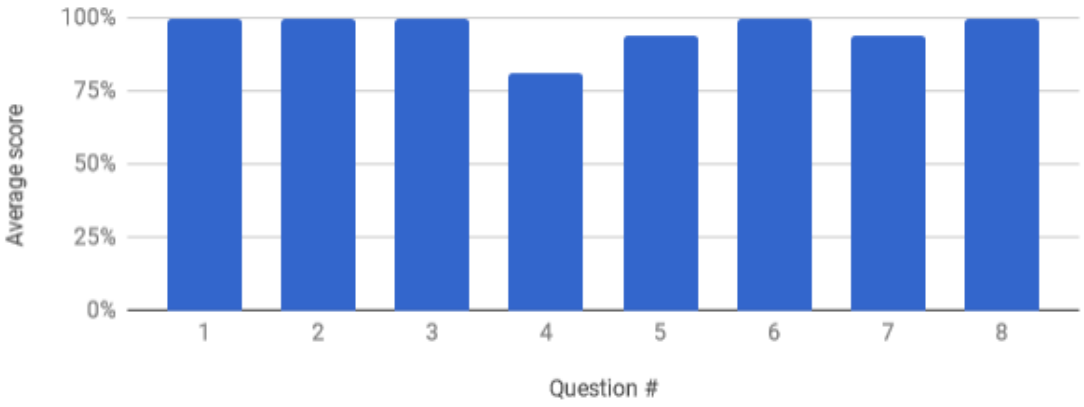

Figure 4. The average score per question of the quiz of the third week

The fourth week of the course covered branching. The quiz was filled by 17 students and had an average score of 84%. The distribution of score is shown in Figure 5. Two students answered all the questions correctly and three questions were answered correctly by everyone.

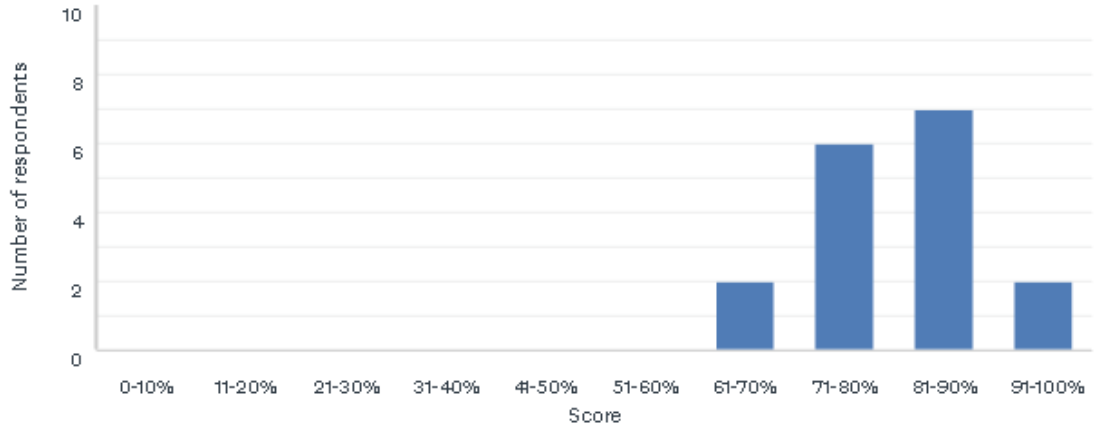

Figure 5. The distribution of scores in the quiz for the fourth week

The hardest question this time was question number seven: "How to sync branches with a remote?". The correct answer was "Branches have to be synced explicitly", but six out of 17 people chose "Branches are synced automatically when you use `git pull` and `git push`". This is an important distinction in how Git handles local and remote changes and evidently should have been stressed more in this week's lecture. However, submissions of this week's practice session and later practice sessions have shown that students had no problem with branching later on. The average score per question can be seen in Figure 6.

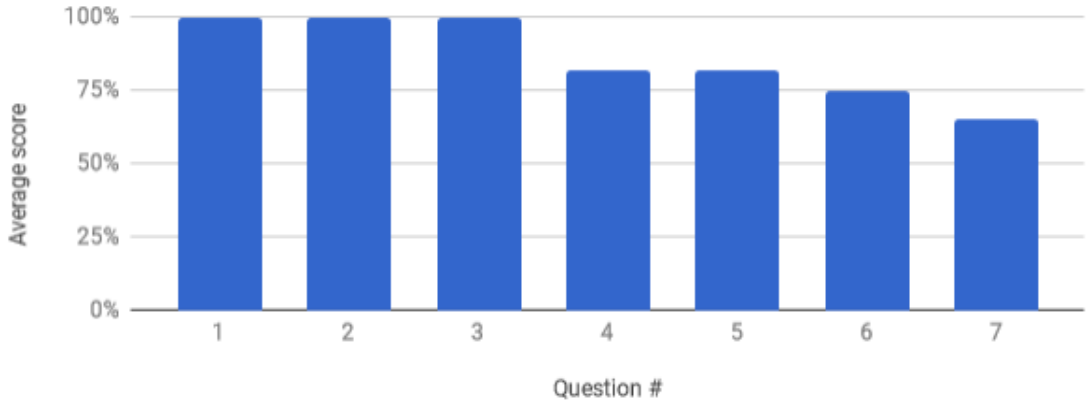

Figure 6. The average score per question of the quiz of the fourth week

The fifth week covered merge conflicts and tracing changes in the code. It was filled by 12 students and the average score was 83%. Two students answered all the questions correctly and three questions were answered correctly by everyone. The distribution of score is shown in Figure 7.

The hardest question was number eight: "What is `git bisect` tool used for?". Six out of 12 students chose the correct answer: "Performing a binary search on the history of a Git repository". Four students chose "Splitting a file into changes made in different commits", one student chose "Splitting a file into changes made in different branches", and one student chose "Performing a binary search on the contents of a file". The second hardest question was number seven: "What command is used to annotate each line of a file with what commit last affected it?". The correct answer was "`git blame`" and as chosen by seven out of 12 students, but four students chose "`git bisect`" and one student chose "`git annotate`". The `git bisect` command was introduced in the lecture but can be considered an advanced Git tool and the results of those two questions suggest that the students had problems understanding it, so perhaps a longer or more detailed demonstration and explanation of the tool is needed. The average score per question is shown in Figure 8.

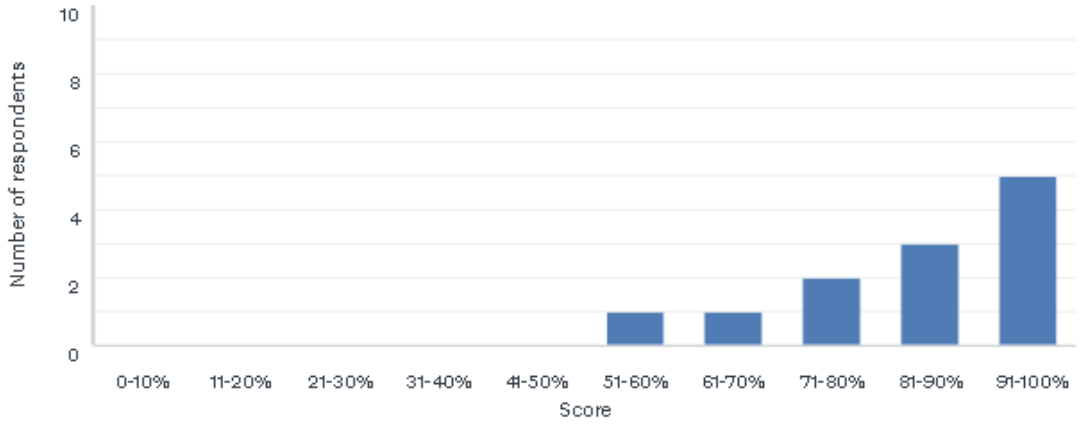

Figure 7. The distribution of scores in the quiz for the fifth week

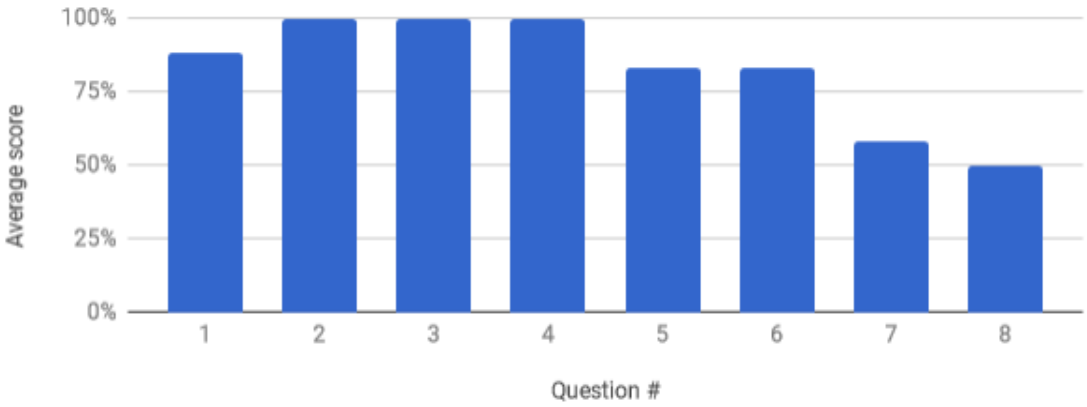

Figure 8. The average score per question of the quiz of the fifth week

The sixth week covered project collaboration tools provided by the GitHub platform. The quiz was filled by 14 students and the average score was 87%. Three students answered all the questions correctly and no question was answered correctly by everyone. The distribution of the score can be seen in Figure 9.

The hardest question was number one: "Which of these is not part of GitHub's project management tools?". Multiple answers could have been chosen, the correct ones being "Velocity calculator" and "Team chat". All students chose the former, but only nine out of 14 students chose the latter. Additionally, one student chose "Wiki". In the case of this question, it seems that most students simply did not notice that it was possible to choose more than one correct answer. The average score per question can be seen in Figure 10.

The seventh week of the course was the last week to have a quiz. It covered the topics

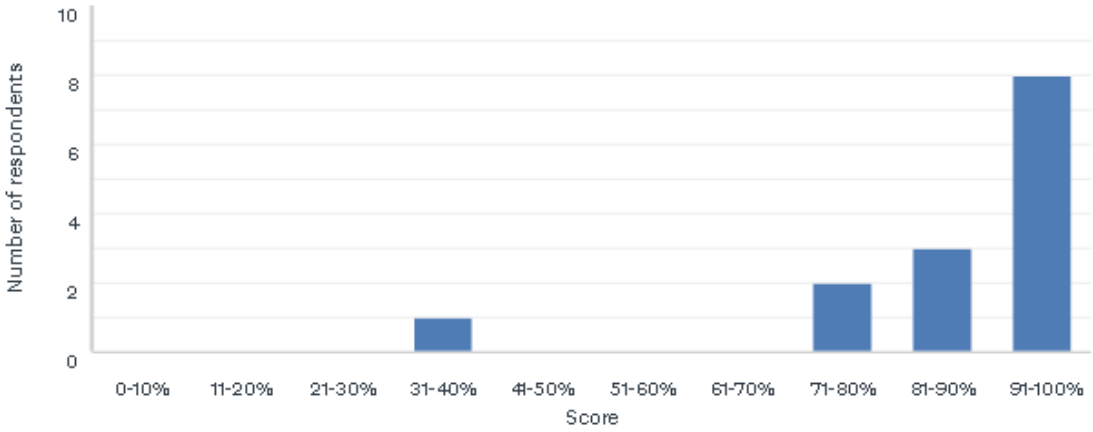

Figure 9. The distribution of scores in the quiz for the sixth week

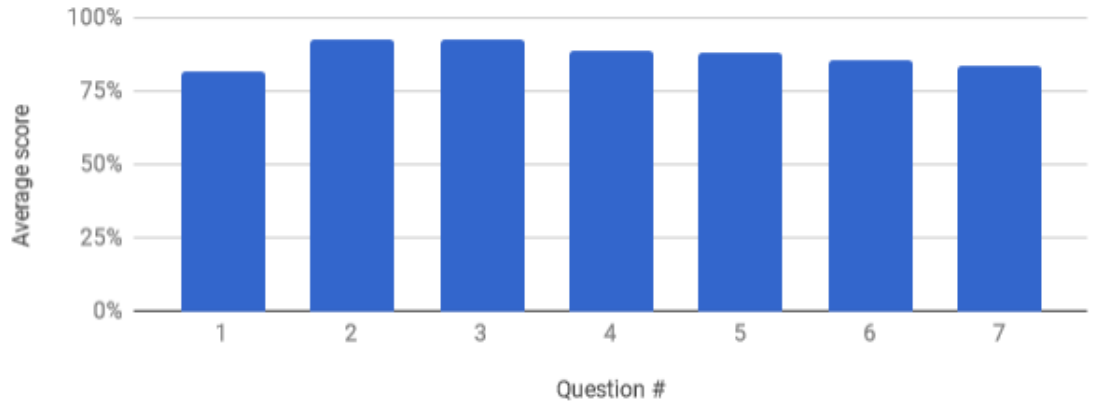

Figure 10. The average score per question of the quiz of the sixth week

of Git hooks and Continuous Integration (CI). The quiz was filled by 14 students and the average score was 94%. Four questions were answered correctly by all the students and seven students answered correctly to all the questions (see Figure 11).

The most incorrect answers were given to question three: "Which is not a category of Git hooks?". The correct answer, "Branching hooks", was chosen by 12 out of 14 students, one person chose "Receiving hooks", and one chose "Other client-side hooks". However, it should be noted that there were only two incorrect responses, which is a low number that does not suggest that the topic of the question was especially difficult. Moreover, the categories were already mentioned explicitly several times during the lecture. Another question with the same average score was question number eight: "How

can CI be used on GitHub?". It had five options and students could choose multiple responses:

1. GitHub has their own CI tool;
2. With receiving (server-side) Git hooks;
3. With a CI application from GitHub Marketplace;
4. With GitHub Webhooks;
5. With a special script added to the GitHub repository.

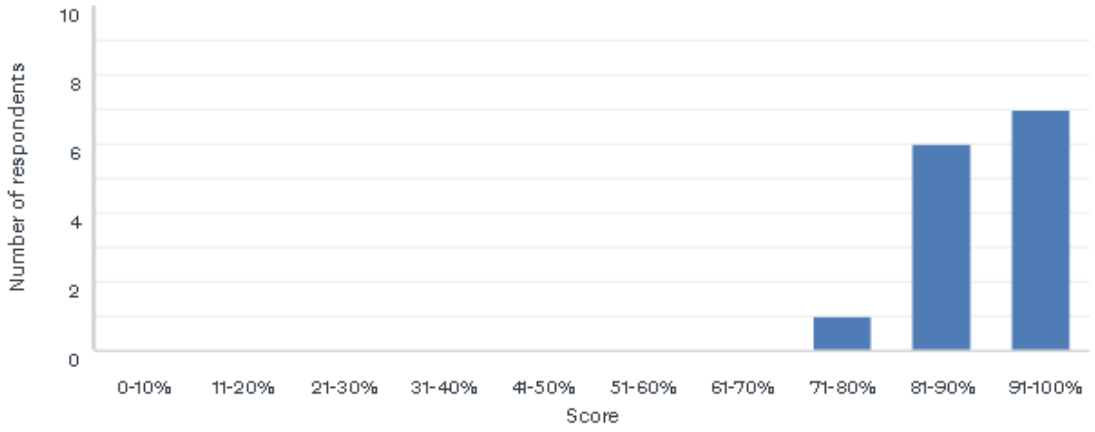

Figure 11. The distribution of scores in the quiz for the seventh week

Out of those, options three and four were correct, but other options were also popular: option two was chosen by eight out of 14 students. The results of this question suggest that the students might have gotten confused by the different ways of setting up CI in GitHub and with other repository hosting options and the different options should have been explained in a more approachable way. In general, this quiz was answered quite well and the average scores of different questions were quite similar (see Figure 12). Several students mentioned in the feedback that they were glad that CI was explained in the course, as they were curious about it and considered it a good introduction to the technology that they would have to use in the Software Project course in the next semester.

In conclusion, the quiz results show that the course was successful in enabling students to achieve a good understanding and mastery of the knowledge and skills that were covered in the lectures and practice sessions. Still, some of the mistakes were

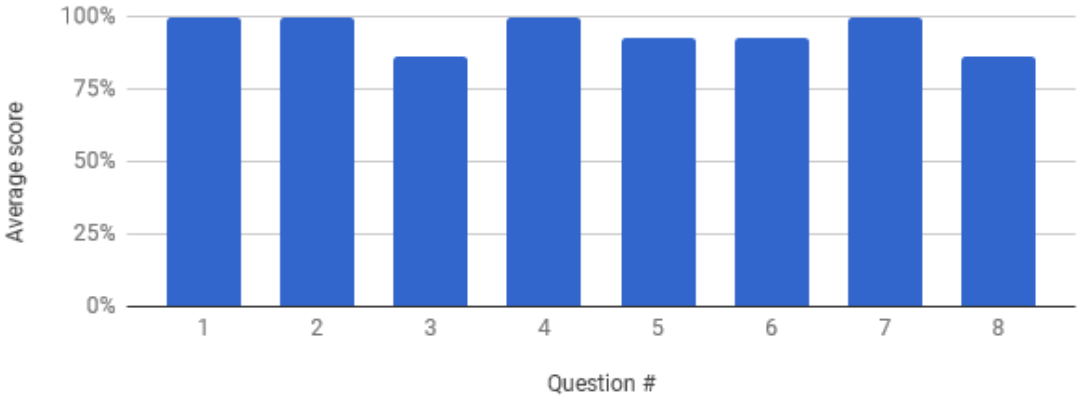

Figure 12. The average score per question of the quiz of the seventh week

common enough to suggest that certain topics could have been handled better and their presentation should be revised for a potential future iteration of the course. The following topics seemed to have caused the most confusion:

- Different files holding Git configuration parameters and ways of changing the configuration;
- The differences between the `git rebase` and `git commit --amend` commands;
- The lack of automatic propagation of changes (e.g. new branches) in Git;
- The usage of the `git bisect` command;
- Different ways of setting up CI.

4.2.2 Feedback questionnaire

The final course feedback questionnaire was created in order to collect the students' opinion on the course. It was filled during the practice session of the seventh week of the course, which was attended by 17 out of 24 students. The structure of the questionnaire is presented in Table 1. The results of the questionnaire show that the students were largely satisfied with the organization of the course (Figure 19), the presentation of the materials (Figure 15), the help that they got from the instructor (Figure 14), and the overall effectiveness of the instructor's teaching (Figure 18). The clarity of the explanations of the concepts (Figure 13), clarity of the instructions (Figure 20), and feedback from the instructor (Figure 16) received lower grades but were still mostly well-received. More students disagreed with the statement that the instructor encouraged

student question and participation (Figure 17), which is understandable since the practice sessions were almost always individual and the lectures were mostly focused on the instructor presenting the concepts, not discussion. Overall, the students were satisfied with the course: 71% of the students gave the course a rating of five and the remaining 29% gave it a rating of four out of five (Figure 23). Over half of the students strongly agreed that the course was useful in developing their knowledge and skills on the subject (Figure 21) and their ability to apply theory to practice (Figure 22).

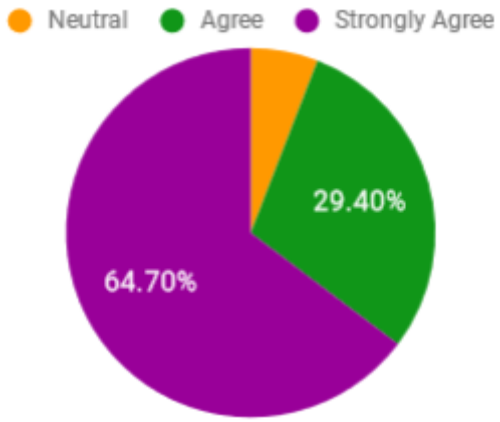

Figure 13. The distribution of evaluation of the statement "The instructor explained the concepts clearly"

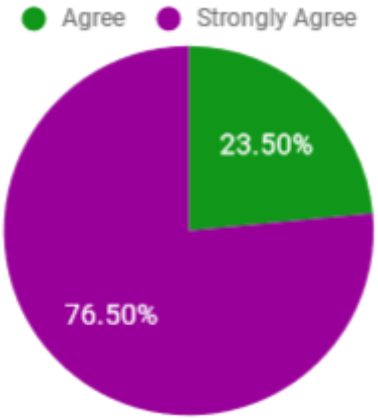

Figure 14. The distribution of evaluation of the statement "The instructor was helpful when I had difficulties or questions"

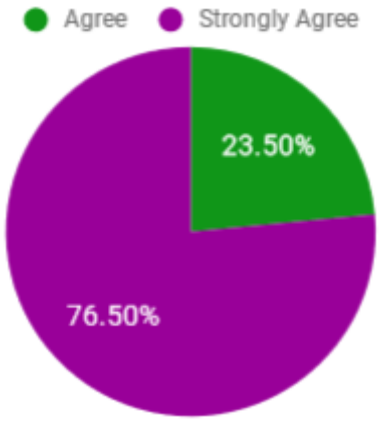

Figure 15. The distribution of evaluation of the statement "The instructor presented content in an organized manner"

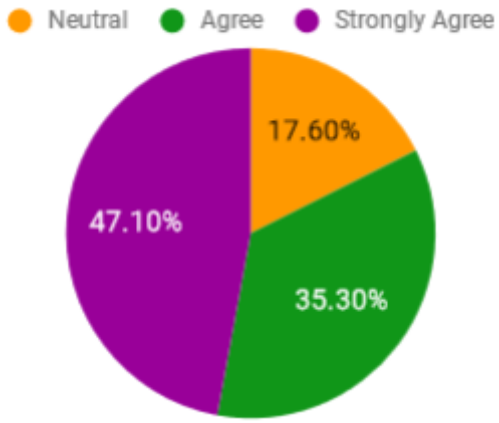

Figure 16. The distribution of evaluation of the statement "The instructor provided clear constructive feedback"

The questionnaire also had three open-ended questions, which asked the students to identify the strengths and weaknesses of the course and leave general feedback. As the strong points of the course, the clear organizational structure and the practicality of the course stand out. As weak points, students answered that some concepts should have been explained more clearly and that the overall structure of the course would be better if it happened every week and so lasted half a semester, instead of having a session every two weeks and lasting the full semester.

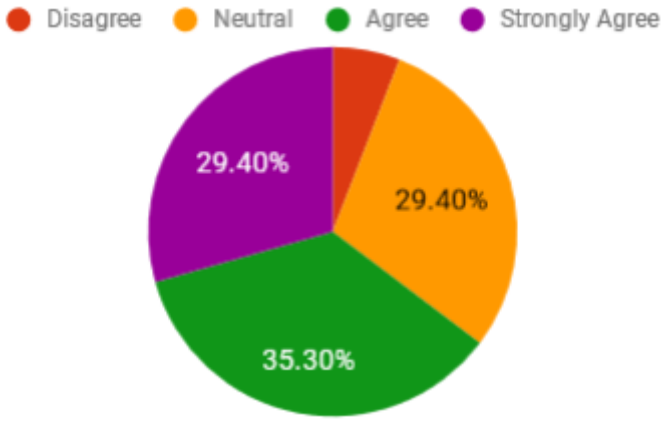

Figure 17. The distribution of evaluation of the statement "The instructor encouraged student questions and participation"

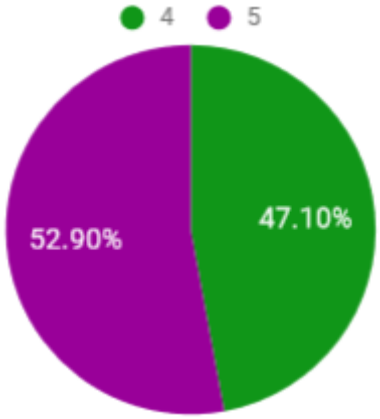

Figure 18. The distribution of answers to the question "How would you rate the overall effectiveness of the instructor's teaching?"

4.3 Ideas for improvement

The results of the weekly quizzes suggest that some of the topics that were presented in the course were somewhat confusing and should have been handled better. Below are some observations on the difficult topics and ideas how their delivery can be improved.

The different files holding Git configuration parameters and the different ways of changing the configuration were the most confusing part of the second week's lecture. The differences should be emphasised and explained more clearly. It is worth considering to revise this part altogether and perhaps simplify it to cover only the global configuration,

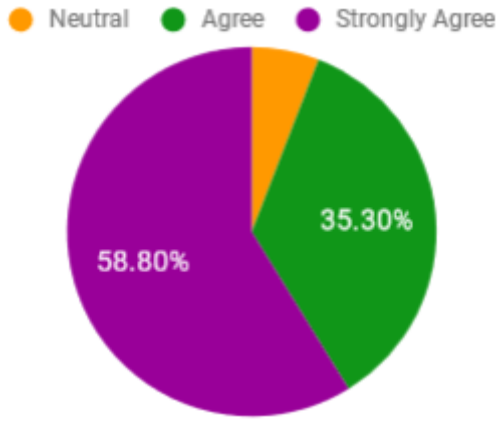

Figure 19. The distribution of evaluation of the statement "The course was effectively organized"

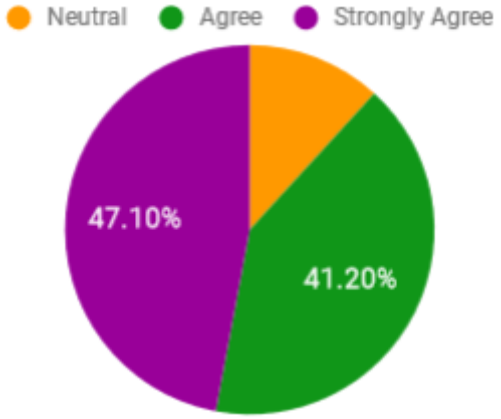

Figure 20. The distribution of evaluation of the statement "The course instructions (including, manuals, handouts, etc.) were clear"

which is sufficient for the basic usage of Git.

The students confused the `git rebase` command with the `git commit --amend` command. While the commands are similar in the way that they can both be used to overwrite commits, their usage is different and they should not be confused. Thus, a clearer explanation is needed. Moreover, only one of the potential uses of the `git rebase` command was demonstrated. During the lecture, it was suggested that rebasing already pushed changes is harmful. In a future iteration, it might be beneficial to demonstrate and explain the potential benefits of a more frequent use of the `git rebase` command for cre-

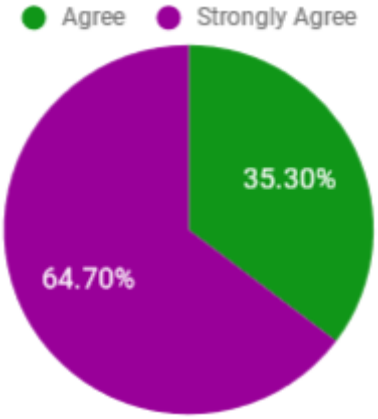

Figure 21. The distribution of evaluation of the statement "The course developed my abilities and skills for the subject"

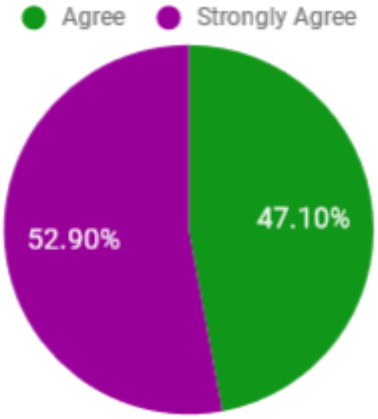

Figure 22. The distribution of evaluation of the statement "The course developed my ability to apply theory to practice"

ating a cleaner commit history [Sia17] and as an alternative to merging of branches[Atla].

The quiz that focused on branching demonstrated that students were expecting branches to be propagated to remote repositories automatically. While everyone seemed to master branching later on through its use in practice sessions, it is important to understand that local-first is an approach that Git uses almost in every part of the system, i.e. all changes are only kept locally unless explicitly specified to be propagated to other repositories by the user. This aspect should have been explained earlier in the course and referenced throughout the semester.

Students showed confusion with the usage of the `git bisect` command. While it

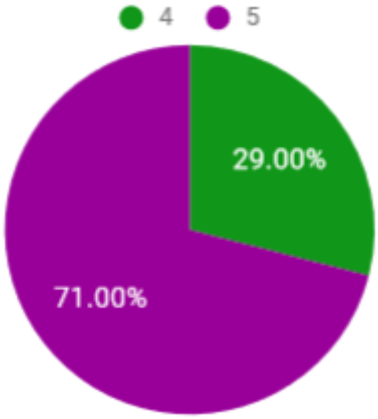

Figure 23. The distribution of answers to the question "How satisfied were you with the course?"

is a powerful tool, it might not be connected enough to the rest of the course to justify covering it. In a future iteration, the importance of the tool and the way of presenting it should be revised.

Lastly, students showed confusion between the various ways of setting up CI. The structure of the lecture should be reconsidered, either providing a more direct and clear explanation of the different ways or omitting some information for the sake of clarity.

Some additional ideas for future improvement came from the students' answers to the final course feedback questionnaire. Some students felt that the course would be more efficient if lectures happened every week instead of once every two weeks. A few also said that they expected more work or more complicated exercises and one student explicitly suggested that the amount of work that was required for the course was less than can be expected of a 3 ECTS course. Another weak point that was identified was that the course wasn't interactive or collaborative enough. Based on these observations, a future iteration of the course could be redesigned to include one or more graded homework assignments in addition to the in-class sessions. For example, the students could be divided into groups and required to work together on the same repository throughout the practical sessions of the course, applying new tools and skills to the same repository every time. This would help reinforce the concepts that are covered in the lectures and practice sessions in a more realistic setting and highlight the connection between the presented concepts and inner-team collaboration as well as demonstrate the importance of solving conflicts that arise from multiple people working with the same repository.

5 Conclusion

This thesis described the process of development and delivery of a new course in the Institute of Computer Science of the University of Tartu. The course, titled Collaboration Tools in Software Engineering, was designed with the European Credit Transfer and Accumulation System (ECTS) score of 3 ECTS in mind and targeted the students of the Computer Science bachelor's curriculum. The course was taught in the spring semester of the 2017/18 academic year and was attended by 24 students.

In Section 2, the learning objectives set for the course were explained. A number of existing courses in the University of Tartu were inspected and demonstrated to refer to or rely on the knowledge of the subject of this course, showing the need for the course. Existing materials were reviewed and shown to not match the set learning objectives. Didactic and design considerations, including schedule and grading scheme, were explained.

A number of materials were created as part of this thesis. In Section 3, the materials were explained in greater detail. The course materials include a set of presentation slide decks for each lecture, video lectures, practice session guidelines, self-evaluation quizzes, and an exam. For each week of the course, the topic was explained, giving detailed insight into the concepts, techniques, and tools that were covered.

Lastly, in Section 4, the materials that were created and used to evaluate the course were explained. The course was evaluated with weekly quizzes and a final feedback questionnaire, which were filled by the students. The analysis of the evaluation of the course shows that the course was useful to the students for getting to know the concepts of Version Control Systems (VCS), issue tracking and collaboration, and Continuous Integration (CI) and that students were mostly pleased with the organization and delivery of the course. The strengths and weaknesses of the course that were highlighted by the students are identified and discussed and improvements for a potential future iteration of the course are proposed.

All of the materials that were created for the course can be accessed from the website of the course in the courses environment of the Institute of Computer Science of the University of Tartu at the following web address: `https://courses.cs.ut.ee/2018/cse`.

References

[Agi] Õppeaine üldandmed: MTAT.03.295 Väle tarkvaraarendus. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1223429+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[Apa] Apache Software Foundation. Subversion Basics. https://openoffice.apache.org/svn-basics.html. Online; accessed 11.05.2018.

[Atla] Atlassian Corporation Plc. Merging vs. Rebasing. https://www.atlassian.com/git/tutorials/merging-vs-rebasing. Online; accessed 19.05.2018.

[Atlb] Atlassian Corporation Plc. Version Control with Git | Coursera. https://www.coursera.org/learn/version-control-with-git. Online; accessed 09.05.2018.

[Aut] Õppeaine üldandmed: LTAT.03.006 Automaadid, keeled ja translaatorid. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1228865+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[BLP] Arne Babenhauserheide, Steve Losh, and David Soria Parra. Mercurial SCM. https://www.mercurial-scm.org/. Online; accessed 10.05.2018.

[Bol] Stepan Bolotnikov. Collaboration Tools is Software Engineering - Courses - Institute of Computer Science. https://courses.cs.ut.ee/2018/cse/spring. Online; accessed 10.05.2018.

[Bol18] Stepan Bolotnikov. Exam | Collaboration Tools in Software Engineering. https://courses.cs.ut.ee/2018/cse/Main/Exam, 2018. Online; accessed 20.05.2018.

[BP16] Matt Butler and Paige Paquette. *Better Software & Stronger Teams*. ZenHub, 2016.

[Cod] Codecademy Inc. Learn Git | Codecademy. https://www.codecademy.com/learn/learn-git. Online; accessed 09.05.2018.

[Col] Õppeaine üldandmed: LTAT.05.009 Koostöövahendid tarkvaraarenduses. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1229547+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[CS14] Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

[dAS09] Brian de Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 36–39, 2009.

[Dri10] Vincent Driessen. A successful Git branching model. http://nvie.com/posts/a-successful-git-branching-model, 2010.

[Ent] Õppeaine üldandmed: MTAT.03.229 Ettevõttesüsteemide integreerimine. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1200832+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[flo16] flow.ci. GitHub vs. Bitbucket vs. GitLab vs. Coding. https://medium.com/flow-ci/github-vs-bitbucket-vs-gitlab-vs-coding-7cf2b43888a1, September 2016. Online; accessed 09.05.2018.

[Gita] Git - Reference. https://git-scm.com/docs. Online; accessed.

[Gitb] GitHub Inc. Features · Project management. https://github.com/features/project-management. Online; accessed 12.05.2018.

[Gitc] GitHub Inc. GitHub Guides. https://guides.github.com/. Online; accessed 09.05.2018.

[Gitd] Github Inc. The world's leading software development platform - GitHub. https://github.com/. Online; accessed 09.05.2018.

[Gite] GitLab B.V. GitLab University | GitLab. https://docs.gitlab.com/ee/university/. Online; accessed 09.05.2018.

[Goo] Google LLC. Google Slides - create and edit presentations online, for free. https://www.google.com/slides/about/. Online; accessed 10.05.2018.

[Inf] Infinite Skills Inc. Learning Git - A Beginners Git Course From Infinite Skills | Udemy. https://www.udemy.com/learning-git/. Online; accessed 09.05.2018.

[Int] Interactive Fate. http://www.interactivefate.com/. Online; acceessed 18.05.2018.

[Jet] JetBrains s.r.o. Version control with IDEA - Help | IntelliJ IDEA. https://www.jetbrains.com/help/idea/version-control-integration.html. Online; accessed 12.05.2018.

[Mar09] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[Mob] Õppeaine üldandmed: MTAT.03.262 Mobiilirakenduste loomine. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1211989+HTML+3081617200650389684 9+text/html. Online; accessed 09..05.2018.

[Obj] Õppeaine üldandmed: LTAT.03.003 Objekt-orienteeritud programmeerimine. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1228862+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[Ope] OpenClassrooms. Manage your code with Git and GitHub - OpenClassrooms. https://openclassrooms.com/courses/manage-your-code-with-git-and-github. Online; accessed 09.05.2018.

[Pan] Collaboration Tools in Software Engineering (LTAT.09.005): Panopto. https://panopto.ut.ee/Panopto/Pages/Sessions/List.aspx#folderID=%226b0d2e71-2455-4315-bdec-6d5da75b8f9a%22&folderSets=3. Online; accessed 10.05.2018.

[Pro] ProProfs. ProProfs - Knowledge Management Software. https://www.proprofs.com/. Online; accessed 13.05.2018.

[Rho] RhodeCode Enterprise. Version Control Systems Popularity in 2016. https://rhodecode.com/insights/version-control-systems-2016. Online; accessed 09.05.2018.

[RVPV09] Siret Rutiku, Aune Valk, Einike Pilli, and Kätlin Vanari. Õppekava arendamise juhendmaterjal, 2009.

[San10] Pablo Santos. The version control timeline. http://blog.plasticscm.com/2010/11/version-control-timeline.html, November 2010. Online; accessed 11.05.2018.

[SC] Jason Long Scott Chacon. Git. https://git-scm.com/. Online; accessed 09.05.2018.

[Sia17] Adrien Siami. Editing your git history with rebase for cleaner pull requests. https://drivy.engineering/git-rebase-edit-history/, April 2017. Online; accessed 19.05.2018.

[Sin11] Eric Sink. *Version control by example*, volume 20011. Pyrenean Gold Press Champaign, IL, 2011.

[Sofa] Õppeaine üldandmed: LTAT.05.003 Tarkvaratehnika. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1228873+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[Sofb] Õppeaine üldandmed: LTAT.05.005 Tarkvaraprojekt. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1228875+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[Sun97] Sun Microsystems, Inc. *Programming Utilities Guide*. 1997. Online; accessed 11.05.2018.

[Sur] SurveyMonkey. SurveyMonkey: The World's Most Popular Online Survey Tool. https://www.surveymonkey.com/dashboard/. Online; accessed 10.05.2018.

[Tho08] Patrick Thomson. Git vs. Mercurial: Please Relax. https://importantshock.wordpress.com/2008/08/07/git-vs-mercurial/, 2008.

[Tra] Travis CI, GmbH. Travis CI - Test and Deploy with Confidence. https://travis-ci.com/. Online; accessed 12.05.2018.

[Uda] Udacity Inc. How to Use Git and GitHub | Udacity. https://eu.udacity.com/course/how-to-use-git-and-github--ud775. Online; accessed 09.05.2018.

[Uni] University of California, Berkeley. Course Evaluations Question Bank | Center for Teaching & Learning. https://teaching.berkeley.edu/course-evaluations-question-bank. Online; accessed 13.05.2018.

[Web] Õppeaine üldandmed: LTAT.05.004 Veebirakenduste loomine. https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1228874+HTML+3081617200650389684 9+text/html. Online; accessed 09.05.2018.

[Yot] Petko Yotov. PmWiki. https://www.pmwiki.org/. Online; accessed 10.05.2018.

Appendix

I. Quizzes

Below, the quizzes that were created for course evaluation are presented. For each quiz, questions are listed. Under each question, it's type and a list of possible answers (where applicable) are presented. Answers that are considered correct are highlighted in bold.

Week 2

1. What is NOT one of the ways to change Git configuration?

Question type: multiple choice.

1. Manually edit the `/.gitconfig` file
2. **Manually edit the `.gitconfig` file in the repository**
3. Manually edit the `/etc/gitconfig` file
4. Use the "`git config`" command
5. Manually edit the `.git/config` file in the repository

2. Which of these is used to create the internals of a `.git` directory in the current directory?

Question type: multiple choice.

1. `git clone --bare`
2. `git init`
3. **`git init --bare`**
4. `git add --bare`
5. `git create internals .`

3. What is the correct form of the command to add a new remote repository?

Question type: multiple choice.

1. **`git remote add <name> <address>`**
2. `git remote add <address> <name>`
3. `git remote add <address>`
4. `git remote <name> <address>`
5. `git config remote.<name> <address>`

4. What does the command "`git clone`" do?

Question type: multiple choice.

1. Copies a file to a given location
2. Gets the latest state of the repository
3. **Copies the entire repository**
4. Stages a file for commit
5. Commits a snapshot of the current working tree to the repository

5. What is "`git add`" command NOT used for?

Question type: multiple choice.

1. Adding a previously untracked file to the repository
2. Adding a modified file to the staging area
3. Adding the contents of a directory to the staging area
4. **Adding a file to the remote repository**
5. Adding a part of a modified file to the staging area

6. What common feature of a modern VCS does Git NOT have?

Question type: multiple choice.

1. **Explicit tracking of file movement**
2. Branching capabilities
3. Distributed workflow
4. Multi-user support
5. Git has all these features

7. Which of these is a correct rule for a `.gitignore` file?

Question type: multiple choice.

1. `.DS_Store`
2. `*.exe`
3. `images/**/*.svg`
4. `build/`
5. **All of these are correct**

8. What is the difference between `git reset` and `git rm` commands?

Question type: multiple choice.

1. `git rm` is an alias for `git reset`
2. **`git rm` removes files from tracking; `git reset` removes changes from the staging area**
3. `git rm` removes changes from the staging area; `git reset` removes files from tracking
4. `git rm` removes files; `git reset` restarts Git
5. "Reset" is a SVN command, not a Git command

9. Any comments on the content and presentation of this lesson:

Question type: open-ended.

Week 3

1. What identifies a commit in Git?

Question type: multiple choice.

1. **SHA-1 hash**
2. MD5 hash
3. Commit number
4. Commit name
5. Secure title

2. Which one of these is NOT associated with a commit

Question type: multiple choice.

1. Commit hash
2. Author name
3. Commit time
4. **Author IP address**
5. Committer name

3. What command is used to view the full commit history?

Question type: multiple choice.

1. `git commits`
2. `git history`
3. **`git log`**
4. `git show`
5. `git rebase`

4. What command is used to change the last commit?

Question type: multiple choice.

1. `git commit --redo`
2. `git commit --rebase`
3. `git commit --change`
4. **`git commit --amend`**
5. `git show`

5. Which command produces the least output per commit?

Question type: multiple choice.

1. `git log --pretty=fuller`
2. **`git log --oneline`**
3. `git log --pretty=full`
4. `git log`
5. `git log --pretty=medium`

6. What command can be used to change commit history?

Question type: multiple choice.

1. `git redo`
2. `git remake`
3. `git recommit`
4. `git checkin`
5. **`git rebase`**

7. Which statement is true?

Question type: multiple choice.

1. You can't overwrite commits in a remote repository because it's impossible in Git
2. **You shouldn't overwrite commits in a remote repository because you might overwrite other people's work**
3. You shouldn't overwrite commits in a remote repository because it's computationally intensive
4. Overwriting commits in a remote repository is allowed and in many cases encouraged
5. Git saves the commits in a remote repository in a backup in case you need them after overwriting them

8. How to restore a file that was previously deleted?

Question type: multiple choice.

1. **`git checkout <commit> <file>`**
2. `git undelete <file> <commit>`
3. `git add <file>`
4. `git rebase <file>`
5. `git pull --deleted <file>`

9. Any comments on the content and presentation of this lesson:

Question type: open-ended.

Week 4

1. What command is used to list existing branches?

Question type: multiple choice.

1. `git branches`
2. `git list-branch`

3. **`git branch`**
4. `git tree`
5. `git log branches`

2. What command is used to switch branches?

Question type: multiple choice.

1. `git branch`
2. `git pull`
3. `git goto`
4. **`git checkout`**
5. `git tree`

3. How does Git manage branches?

Question type: multiple choice.

1. Branches are separate directories
2. **Branches are pointers to certain commits**
3. Branches are separate repositories
4. Branches are special commits
5. Branches are special remotes

4. What is the best name for a topic branch?

Question type: multiple choice.

1. branch_12
2. login_page
3. feature/login_page
4. create_login_page
5. **feature/12_login-page**

5. What is the purpose of a release branch?

Question type: multiple choice.

1. **To prepare a release and make last moment changes before merging into master and releasing**
2. To separate code of the version from past and future versions
3. To create feature branches from it for the development of that version
4. To keep an archive of releases
5. To share the release code to other collaborators

6. What are the acceptable ways of adding new code to the project (choose several)

Question type: multiple choice.

1. Make commits to master branch
2. Make commits to development branch
3. **Make commits to a topic branch**
4. **Make commits to a hotfix branch**
5. **Make commits to a release branch**

7. How to sync branches with a remote?

Question type: multiple choice.

1. Branches are synced automatically when you use `git pull` and `git push`
2. **Branches have to be synced explicitly**

8. Any comments on the content and presentation of this lesson:

Question type: open-ended.

Week 5

1. Which of the situations can result in a merge conflict? (choose several)

Question type: multiple choice.

1. Adding files
2. Making changes to different areas of the same file
3. **Making changes to the same part of the same file**
4. Changing different files
5. **Changing whitespace in a file while someone else is working on it**

2. What happens when Git encounters a merge conflict?

Question type: multiple choice.

1. Git denies merging the two branches
2. **Git pauses merge and asks you to resolve conflicts**
3. Git automatically resolves conflicts, accepting the latest changes
4. Git automatically resolves conflicts, accepting the largest changeset
5. Git automatically resolves conflicts, accepting all the changes

3. What command can be used to stop merge process?

Question type: multiple choice.

1. `git merge stop`
2. `git merge reset`
3. `git merge abort`
4. **`git merge --abort`**
5. `git merge --stop`

4. What happens to conflicted files?

Question type: multiple choice.

1. Git creates new files for "ours" and "theirs" versions
2. Git leaves files as they were before merge
3. **Git adds conflict-resolution markers to the files**
4. Git checks out files from the branch being merged
5. Git checks out files as they were when the history diverged

5. What information do conflict-resolution markers show with `"conflictstyle=diff3"` option? (choose several)

Question type: multiple choice.

1. **The changes that were made in the current branch**
2. **The changes that were made in the branch being merged**
3. **The file as it was before histories diverged**
4. The file at the beginning of the repository
5. The changes that Git proposes for merge

6. What information do conflict-resolution markers show with `"conflictstyle=merge"` option? (choose several)

Question type: multiple choice.

1. **The changes that were made in the current branch**
2. **The changes that were made in the branch being merged**
3. The file as it was before histories diverged
4. The file at the beginning of the repository
5. The changes that Git proposes for merge

7. What command is used to annotate each line of a file with what commit last affected it?

Question type: multiple choice.

1. `git annotate`
2. `git log-file`
3. `git bisect`
4. `git merge`
5. **`git blame`**

8. What is the `git bisect` tool used for?

Question type: multiple choice.

1. Splitting a file into changes made in different commits
2. Splitting a file into changes made in different branches
3. **Performing a binary search on the history of a Git repository**
4. Performing a binary search on the contents of a file
5. Splitting a file into changes made by different authors

9. Any comments on the content and presentation of this lesson:

Question type: open-ended.

Week 6

1. Which of these is not part of GitHub's project management tools? (select several)

Question type: multiple choice.

1. Issue tracker
2. Project boards
3. **Velocity calculator**
4. **Team chat**
5. Wiki

2. Which of these is a valid user story?

Question type: multiple choice.

1. Upload button must open video upload dialog
2. Implement video uploading
3. Video uploading currently fails
4. **As a content creator, I want to be able to upload a video to reach my audience**
5. Users need a way to upload video

3. What are the three levels of project management solutions?

Question type: multiple choice.

1. integrated application; stand-alone repository hosting solution; mobile application
2. first-; second- and third-party applications
3. first-; second- and third-party applications
4. web application; desktop application; mobile application
5. **All-in-one hosting and project management; Integration over existing solution; stand-alone application**

4. Which of these are required data for an issue? (select multiple)

Question type: multiple choice.

1. **ID number**
2. **Title**
3. Description
4. Milestone
5. Assignee

5. How can you close an issue in GitHub? (choose several)

Question type: multiple choice.

1. **On GitHub website**
2. **From a commit message**
3. **By merging a related pull request**
4. By e-mail
5. From commited code

6. Which of these cannot be a card on GitHub project boards?

Question type: multiple choice.

1. Issue
2. **File**
3. Note
4. Pull request

7. What can you do as part of code review? (choose several)

Question type: multiple choice.

1. **Approve changes**
2. **Leave comments without approving or disapproving**
3. **Disapprove changes**
4. **Comment on specific lines of code**
5. Make changes to code

8. Any comments on the contents and presentation of this lesson

Question type: open-ended.

Week 7

1. What is Continuous Integration?

Question type: multiple choice.

1. The practice of writing automatic tests
2. Deploying the software every time a change is made
3. **Automatic testing and building every time code is changed**
4. Pushing to the remote every time a commit is made
5. Doing work in separate branches

2. What is Continuous Deployment?

Question type: multiple choice.

1. Pushing to the remote every time a commit is made
2. **Making the latest stable version of software available at all times**
3. Doing all work in master branch
4. Writing automatic tests
5. Having more than one remote in your Git repository

3. Which is not a category of Git hooks

Question type: multiple choice.

1. Commit hooks
2. Receiving hooks
3. Email workflow hooks
4. Other client-side hooks
5. **Branching hooks**

4. What language are Git hooks written in?

Question type: multiple choice.

1. C/C++
2. Java
3. Gitscript
4. bash
5. **Any programming language**

5. Where are Git hooks located?

Question type: multiple choice.

1. In the working directory
2. In the `.git` directory
3. **In the `.git/hooks` directory**
4. In the hooks directory of the working directory
5. In the `.githooks` file

6. Can you make a Git hook on the server that would not allow anyone to do a `git push` to that remote?

Question type: multiple choice.

1. **Yes**
2. No

7. What is the correct filename of a "pre-commit" Git hook written in the Python programming language?

Question type: multiple choice.

1. `pre-commit-hook.py`
2. `pre-commit.hook`

3. `pre-commit.py`
4. **`pre-commit`**
5. `hook_pre-commit`

8. How can CI be used on GitHub? (Choose several)

Question type: multiple choice.

1. GitHub has their own CI tool
2. With receiving (server-side) Git hooks
3. **With a CI application from GitHub Marketplace**
4. **With GitHub Webhooks**
5. With a special script added to the GitHub repository

9. Any comments on the contents and presentation of this lesson

Question type: open-ended.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Stepan Bolotnikov**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis

   Development of a Course on Collaboration Tools in Software Engineering

   supervised by Marlon Dumas

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 21.05.2018