Using the *Git* Version Control System to Replace a Learning Management System

Gunnar Wolf, *Graduate Student Member, IEEE*

***Abstract*— For several years now, but with special force since the beginning of the COVID contingency, higher education professors have turned to telematic tools that help teachers manage the work with which their students practice and demonstrate their progress in class – Tasks, practices, projects, exhibitions and more. The systems that, over the last two decades, have been most used for this purpose are called Learning Management Systems (LMS); The best known is undoubtedly Moodle, but there are a large number of options for this. Based on the empirical observation of a rejection by students of the use of these systems, the authors made the decision to completely replace their use with that of a version control system widely used for software development, *Git*, specifically for teaching students of the Computer Engineering career, for the Operating Systems course. This article presents the justification for choosing this platform, and the evaluation of the experience after eight semesters using it.**

***Index Terms*— Computer applications, learning technologies, version control, learning management systems.**

I. Introduction

THE experience presented in this work is located within the teaching of the *Operating Systems* class of *Computer Engineering*, taught at the Faculty of Engineering of the National Autonomous University of Mexico.

The article begins with a concepts review. The present Section presents the concepts of Learning Management Systems (LMS) as well as Version Control Systems (VCSs). Naturally, in the last 30 years, several experiences of using VCSs as part of the workflow of school groups have been documented; Section II documents related works, succinctly contrasting their contributions or differences with the model reported by this work. Section III describes the operation, approach and basic interaction of students with the VCSs. In order to validate the experience as useful or successful in the eyes of the students, a survey was carried out in November 2020, which was answered by 13% of the students. Section IV presents the results of said survey. Finally, Section V presents general conclusions about the reported experience and outlines possible future paths to delve deeper into it.

Manuscript received 21 June 2022; revised 9 August 2023; accepted 2 October 2023. Date of publication 20 February 2024; date of current version 26 March 2024.

This work involved human subjects or animals in its research. The author confirms that all human/animal subject research procedures and protocols are exempt from review board approval.

The author is with Facultad de Ingeniería (FI) and Instituto de Investigaciones Económicas (IIEc), Universidad Nacional Autónoma de México (UNAM), Coyoacán, Ciudad de México 04510, Mexico (e-mail: gwolf@iiec.unam.mx).

A Spanish version of this article is available as supplementary material at https://doi.org/10.1109/RITA.2024.3368293.

Digital Object Identifier 10.1109/RITA.2024.3368293

A. Background: Learning Management Systems (LMS)

Distance learning tools, be they aimed at allowing students to learn at the pace and schedule of their choice, or to deliver teaching to places lacking coverage by a given educational system, are neither a new idea nor a recent development. Before the development of information technologies and global data networks, various learning centers emerged around the world with varied responses; An example of this is correspondence courses or *telesecundarias* [1].

With the massification of Internet access in the 1990s, various systems emerged that, using the Internet, allowed the distribution of educational materials to students following the rhythm and logic of the course, while also allowing them to use the same medium bidirectionally – offering forums to express their doubts, delivery places for assignments and exams. Although this began by adapting simple Content Management Systems (CMS), the possibility of managing administrative aspects of the course (defining courses, enrolling students, registering teachers, defining relationships between them, keeping progress reports and grades), as well as the development of modules for specific interactions with the educational environment, defined specialized systems called *Learning Management Systems* [2].

Given the facilities they provide to teachers, the adoption of LMSes soon exceeded the original approach of being used for distance or mixed education, to become a common tool also for traditional classroom teaching. An LMS can inform students of new assignments or notify them when their work is due. It provides the teacher with a single point of management for digital submissions, replacing the traditional suitcase full of papers to grade. It allows the teacher to design quizzes and exams that students can answer, even outside of class hours – and even allows these exams to be graded automatically, resulting in significant time savings for the teacher [3].

B. Background: Technological Framework for VCSs

Software development, at any serious scale, has always required the coordination of efforts between developers, and it results natural to expect that tools for this purpose have been developed very early in the history of computing. The development model followed until the 1960s was very different, but as the interactive use of computers became popular through time shared systems and interactive terminals, this

need became apparent. The first document found that covers VCSs mentions:

> Because sccs represented such a radical departure from conventional methods for controlling source code, it became clear when we began development of it (in late 1972) that a paper specification would not be sufficient to "sell" the system to the software projects to which it was intended; we would have to have a working prototype. [4]

As the authors assumed, it took several years for the idea to gain the traction needed to become common in software project development; By the middle of the following decade, the main tools for this purpose were *sccs* and *RCS* [5]; towards the end of the eighties, and motivated due to the three participants in a project had incompatible schedules, *CVS* was developed. It was initially a set of helper functions on *RCS*, which facilitated parallel development without explicit coordination [6]. Several years later, it becomes clear: "one way to predict whether a software project will be successful is to ask, do its developers use a version control system to coordinate their work?" [7]

The large free software projects started in the early 1990s (particularly the Linux, FreeBSD, NetBSD and OpenBSD operating systems, and a large number of applications) began their development using *CVS* as a nodal point. It is worth mentioning that, in addition to technical development projects, those mentioned here share a socio-ideological factor: the ideological movement of free software seeks the creation of a body of software that does not impose on its users and potential developers the limitations derived from copyright, and very particularly, that allows source code reappropriation, modification and redistribution [8].

The 1990s witnessed rapid growth both in free software projects and in the computing equipment's capabilities, as well as in and the universality of network access (it can be argued that the first was due to the conjunction of the first two). The *CVS* usage model began to prove insufficient; *CVS* lacked the ability to represent actions as common as deleting or renaming a file, and imposed a high cost when working with non-plain text files.

Subversion was developed between 2000 and 2004; this system followed the same interaction model of *CVS*, correcting these and other weaknesses [9]. *Subversion* grew in a few years and became one of the main development tools in the free software field.

However, all VCSs mentioned up to this point operate centrally. With the growth of some projects to development scales unimaginable until then (8,000 developers in 20 years, 15 million lines of code and more than 37,000 files [10]), in 2002 Linus Torvalds made the controversial decision to leave *CVS* and adopt a *distributed* VCS (DVCSs): a system in which there was no single primary copy or central server, but rather, each copy of the project stored the full history and state, allowing synchronization between related branches. However, since there was no VCS that met the postulates of free software and implemented the distributed model, Torvalds made the pragmatic and controversial decision to adopt *BitKeeper*.

Although politically the adoption of *BitKeeper* was difficult and a recurring friction point with software freedom purists, there is no doubt that it helped tremendously to regain speed in the Linux project development, which had been suffering for several years [11]. *BitKeeper* offered a free (*as in beer*, not *as in freedom*, would be often repeated) license to free software developers, as long as it was not used to compete with *BitKeeper* itself.

By 2005, an argument arose over whether the functionality Andrew Tridgell was adding to Linux was a license violation [12], which eventually led to Torvalds himself initiating development of an DVCS, which he called *Git*.

In the same period of time, several other free DVCSs were developed, such as *Monotone*, *Darcs*, *Mercurial* or *Bazaar*. There are also other proprietary alternatives, such as Microsoft's *Team Foundation Server*. However, *Git* has been a clear winner over the others, and today it can be seen as *lingua franca*, not only among free software developers, but in the world of programming in general; it has brought together not only a clear majority of projects among the competing DVCSs, but has managed to attract a large number of projects that even had more than twenty years of history on centralized VCSs [13].

In parallel to what has already been presented, since the late nineties, various *forges* appeared: websites dedicated to hosting free software projects, to which they provided managed resources, such as a Web space, bug tracker, mailing lists – and an SCV. According to the information available on the first and at some point best known of said forges, *SourceForge*, it currently hosts more than 500,000 projects and has "several million" registered users. However, the interaction flow on which it is based is not very friendly, which makes it suitable only for users who are already software development professionals.

In 2008, and already seeing rapid growth in the adoption of *Git* for projects of all sizes, *GitHub* was born: a forge with a simplified workflow, and strongly focused on the *Git* development model. Just three years later it was already the most active forge in the world of free software [14]. As of the date of this writing, according to the information available on the website, it hosts more than 100 million projects and more than 50 million developers.

Now, although *GitHub* has become nodal for the development of this impressive number of free projects, it causes a certain cognitive dissonance that *GitHub* itself is not free: the software with which the website operates, and which integrates the different tools that make it up, is proprietary. There are other comparable services, such as *GitLab*, which even offers a collaboration model and semantics perfectly mappable against that of *GitHub*. It would be more consistent with the author's personal principles of use and promotion of free software with which the project that our text reports began to develop the experience described below on *GitLab* or some similar platform; The decision to do so on *GitHub* was not taken lightly, and focuses on the importance that this site has for software development in general. Today, virtually all free development projects are hosted on *GitHub* (whether

it is for providing their main development space, or is chosen as backup or convenience storage).

C. About This Document

The work here presented is a re-elaboration and update of a very preliminary version, presented in the 2017 Free Software and Education On-line Encounter (EDUSOL), published in full [15]; the main additions since that version was published are:

- The study is carried out over a much larger period.
- A much deeper review of related works.
- Ellaboration and report of a survey on students, validating several of the presented aspects.

II. Related Work

The use of VCSs as a mechanism for the delivery of coursework is not a new idea. Experience reports can be found as early as in 1989 [16], although in said work, the teacher was unable to complete the implementation due to the students' difficulties in understanding the system. Quoting the author,

> The students were supplied with the machine-readable copy of the source code contained within a revision control system. To the disappointment of the instructors, the revision control system was not used, and greater emphasis will be given to the benefits of such a system in future workshops.

Regrettably, it was not possible to locate information regarding later experiences regarding this early experience.

Several works report experiences in the same general lines as ours, although understandably, starting from different tools, according to the point in time in which they were presented: In all cases, they refer to centralized VCSs [7], [17] [18]. After several years of not finding literature on the matter, a much closer reference to the work presented here can be found in Glassey's text [19].

It is important to emphasize that VCSs do not cover all of the functionality that LMSs present. The experience that this article seeks to show focuses on the component for communicating instructions from the teacher to the students and deliverables in the inverse direction; It does not consider roles such as supervisors, tutors and administrators, nor does it intend to cover functionality such as feedback forums, feedback tools, educational content modules, nor other communication mechanisms, etc. [20, Section 2.1]

A. Coursework Employing Centralized VCSs

Almost all of the experiences reviewed presented the use of a VCS following a model of an independent repository per student. This is antithetical to the expected use of the implementation to work with collaborative development: If each of the students has an independent repository, how are group work submissions managed? Should each team member send the same submission separately to the teacher? Does only one of them commit the deliverable, notifying the professor out of band where to look for the work? When discussing this point, Reid and Wilson even mention an important point that is considered central to the discussion [7]: "preventing students from seeing each other's work." Collaborative development systems are, then, presented in a rather adversarial environment, working against the natural collaboration that they should implement.

Milentijevic's implementation presents an experience built on a technological foundation of *Subversion* or *CVS*, but necessarily mediated by a Web interface [18]. A single repository is not managed throughout the course, but progresses by separating each delivery: throughout the course, a new repository is generated for each task or project; each student must obtain the repository. The deliverables, the authors explain, include the life cycle of each of these developments: Definition and assignment of tasks, work on the task, completion and evaluation. Each of these stages requires a different type of interaction between supervisor and students, involving a nontrivial amount of bureaucratic work (covered by the Web system) for the model presented.

Glassy's proposal contrasts with the previous two, given that their methodology does not seek the use of a VCS as a mechanism for the delivery of tasks and projects, but rather as a tool to evaluate the development practices of their students [17]. Each student creates their *Subversion* repository, performs the work, and submits by email to the entire repository. Glassy analyzes the patterns of *commits* (committed change sets) that are part of the repository, and documents patterns not too different from those that will be presented in Figure 2.

B. Coursework Employing Distributed VCSs

Glassy's *Related Work* section refers to a course at the author's university (Montana Tech) in spring 2005, based on Darcs [17]. This would be the first case of a course on a DVCS, and the experience would be very interesting to analyze, but it was not possible to find more information about it.

GitHub operates a community called *GitHub Education Community*, dedicated to the generation of contact between teachers and students and discussion of the use of DVCS technology for educational purposes. It also offers organizational help for *Git* repositories, through organizations that operate *GitHub* Rooms; The flow presented is specific to the proposed configuration and does not reflect collaborative work in a professional space. Each student works and makes their deliveries in their own repository. *GitHub Classrooms* was originally developed as free software, but was converted to an internal and closed *GitHub* project in early 2020 [21].

Glassey publishes a comparison of tools to control groups, focusing on courses implemented on *Git* that use *GitHub*, but is rather focused on the tools developed by different teachers to facilitate the monitoring of deliveries by students [19]. The work presented here does not have or require external tools for this purpose – although it has only been used with medium groups (up to 40 students), it is the author's opinion that, by far, the greatest burden in time for the teacher is grading of delivered projects, not tracking whether the projects have already been delivered or not.

A particularly relevant part of Glassey's article is the comparison between the distribution models of the different systems: Seven of the eight flows he reviews distribute

assignments to students following the model he calls ORpSpA (*One Repository per Student per Assignment*), and the remaining, OBpP (*One Branch per Problem*). The implementation reviewed here is independent and novel with respect to all the cases in Glassey's review.

III. Implementation

Our Engineering Faculty offers several LMS system facilities, particularly the cluster named *EducaFI*, which has a Moodle for each of the Faculty's divisions, managed by the Academic Computing Services Unit (UNICA).

The teaching experience this work reports begins in January 2013 with the 2013-2 semester. From our first course, a decision was made to use EducaFI as a partial tool for delivering assignments and for relaying related bibliography and other activities that were being covered to the students; The use of Moodle as an LMS was relatively limited, with only two exercises carried out from questionnaires developed with assigned qualification, and without using any additional modules.

In general, the administration by the group of interns at User Service at UNICA is good and agile, but limited to its operating policies; in particular, we requested the courses already taught to be preserved as a memory, in order to be able to compare progress with subsequent experiences. Due to their inability to comply with this, and due to occasional problems of some students with problems managing their accounts, the decision was made to migrate the LMS to another instance of Moodle, managed by this text's writer, and located at a research institute of the same university. It was then considered to add some modules that would be useful for the course, such as GeSHi for the presentation of code fragments or the Virtual Programming Lab (VPL) to offer a development environment where students could solve programming approaches without depending on a specific environment; It is necessary to recognize that this remained a mere intention due to the time required to know and take advantage of the Moodle environment beyond casual use.

Towards the end of the 2016-2 semester, the decision was made to, instead of seeking to offer students tools that *simulate* those that students will find in a realistic software development environment, lead them to actually use them, leading to the adoption of *Git*. After evaluating the points presented in Subsection I-B, in preparation for the 2017-1 semester, a *Git* repository called *clase-sistop-2017-01* was prepared, hosted in the teacher's personal space at *GitHub*, at the https://github.com/gwolf/clase-sistop-2017-01 URL. In later iterations, the URL was simplified to help students find it, and starting with the 2020-1 semester, they were located in the Engineering Faculty organizational space, as can be seen in Table I:

Regarding the internal structure that was given to the repository, at first, it only contemplated three directories of the repository for student submissions (*homework*, *practices* and *projects*), with each of the deliveries located in a numbered subdirectory within these. Shortly after, a directory for presentations was added to this, which although they follow different logic for their delivery, can be framed in the general workflow. Before long, it became obvious that the *Git* repository was also an ideal medium for transmitting content to students – code snippets written as in-class examples, references to websites or other resources mentioned, etc.

TABLE I
URLs for Each of the Semesters' Repositories

2017-2	https://github.com/gwolf/sistop-2017-2
2018-1	https://github.com/gwolf/sistop-2018-1
2018-2	https://github.com/gwolf/sistop-2018-2
2019-1	https://github.com/gwolf/sistop-2019-1
2019-2	https://github.com/gwolf/sistop-2019-2
2020-1	https://github.com/unamfi/sistop-2020-1
2020-2	https://github.com/unamfi/sistop-2020-2

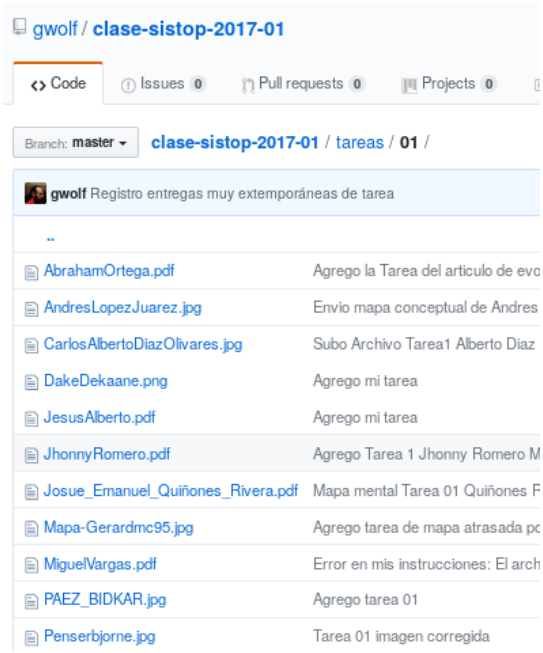

Fig. 1. Example of delivery directory for the first semester where this methodology was applied. note that several students named their files using pseudonyms, not their real names, which made assigning grades correctly difficult.

When presenting our work scheme, in practice 1, students are explained under what assumptions deliveries are structured in the way they are, through the following approach:

> "The course repository is a development project to which you are interested in contributing. When the teacher generates a new activity, an important defect naturally "appears" (a *bug*, in software development terms): your participation is not registered for that activity. What you need to do is provide a patch for that defect. To do so, update your local copy of the project, develop the steps to correct the bug, and send it to the teacher so that he can incorporate it into the main project through a *pull request*."

The first semester where *Git* was used, no naming standard was requested from students for their deliveries, so naturally it became complicated to track the relation between files and the student to which it belonged; this can be appreciated in some of the directories shown in Figure 1.

In addition to this change, the structure of the repositories used over the different semesters has remained basically unchanged.

Given that, in general, students are not familiar with *Git* at the beginning of the course (nor with any other VCS), it was considered necessary to introduce its basic operation through three practices; we explain to students that, for grading purposes, although the submission of assignments is mandatory (a student who does not submit all assignments loses the right to exemption, and a student who does not submit 80% of assignments loses the right to present the final exam in the first round), practices are optional, and their only effect is to improve their final grade.

The practices related to the use of *Git* that are carried out as part of the course are:

1) **Using *Git* and *GitHub***: It consists of eight steps, with complete and detailed, tutorial-style instructions. We ask the student to create an account in *GitHub* if they do not have one, to make a *fork* of the main repository (this is, to copy it into their personal space, where they have modification permissions), clone it to their personal computer, generate a file with their name within the delivery directory of said practice, send it to their *fork* in *GitHub*, and notify the teacher of their changes through a *pull request* (request to incorporate the changes of a *fork* in the main branch).
2) **Parallel development branches**: A central feature of *Git*'s success is that it makes it easy to work on different aspects very efficiently: By using topic branches, a developer can be working on different features independently. The management of branches allows students to begin early to develop their projects and presentations (activities that are intended to be distributed over time) without this preventing them from making minor deliveries, such as other practices or homework.
3) **Deleting unnecessary files**: An important provision of all VCSs is the ability to ignore auto-generated files resulting from the compilation or execution of a program. This practice is included to prevent students from getting confused with the changes in these files, and to guide them towards good practices of collaborative development.

In our experiences so far with *Git* management, class time has also been devoted to explaining the model; although precise measurements were not made, the total time spent is estimated at less than two and a half hours - approximately one class session over the course of the semester. Figure 2 illustrates a snapshot of one of the repositories showing how each student diverges from the main branch of the repository to prepare their submissions, and how these are incorporated back into the main branch; we do invite you to use the public and interactive version of said graph to explore it and better understand the various events it records, from the addresses presented in the I Table, by clicking on *Insights* → *Network*.

Besides the three practices mentioned above and of the explanation made in class, there is nothing in particular characterizing the students' usage of *Git* as classroom-specific: Students perform exactly the same tasks they would in a non-academic project.

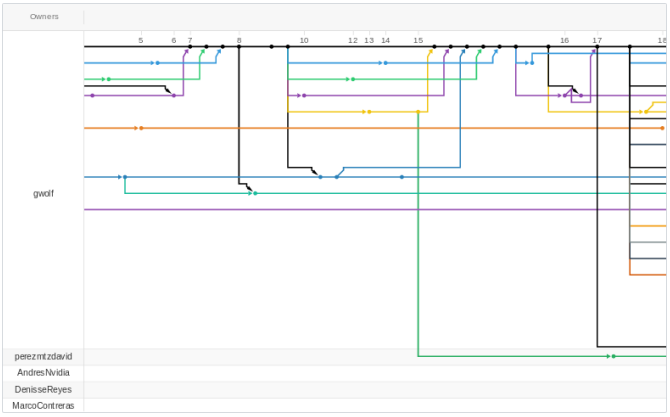

Fig. 2. Network graph for the interaction between repositories for the 2017-1 repository. Each dot represents a *commit*. Some patterns can be observed, indicating deadlines for handing over assignments (days 15-16), announcing a new task to be carried out (day 17), students handling parallel development lines (forking of the blue line around day #10, forking of the yellow line at day 17). Lines towards the bottom represent students that have left some work pending; as they have not yet sent a *pull request* or not having complied with the requested changes in any of them.

TABLE II
COMMITS IN EACH OF THE REPOSITORIES, REPORTED BY SEMESTER

Semester	Students	C_{Tot}	C_{Stdt}	C_{Tch}	$\frac{C_{Stdt}}{Stdt}$	$\frac{C_{Stdt}}{C_{Tot}}$
2017-2	27	316	178	138	6.59	0.56
2018-1	21	473	294	179	14.00	0.62
2018-2	32	331	158	173	4.94	0.48
2019-1	31	599	388	211	12.52	0.65
2019-2	30	964	613	351	20.43	0.64
2020-1	33	1273	970	303	29.39	0.76
2020-2	43	777	528	249	12.28	0.68

IV. RESULTS

Git is not an LMS, nor this work seeks to present its applicability in a generic way as if it were. The experience reported here is necessarily limited to the presented scenario: a course oriented to software development, and a personal commitment of the teacher to train students in the use of collaborative development tools that will most likely be valuable for their professional performance.

The delivery unit in *Git* is the *commit*: When a user determines that the changes they made constitute a *milestone*, they group them under a *commit*; the term means that the user *commits* or that *acquires a commitment* to a set of changes as a coherent unit. To submit an object (practice, assignment, etc.), a student must make an absolute minimum of one *commit*, after which he or she notifies the teacher via a *pull request*; The teacher makes a second *commit* for each of them, incorporating it to the main branch.

Table II shows some data regarding the *commits* made in the repositories of the seven semesters reported in this work. The "Students" column indicates the total number of students enrolled in the course. C_{Tot} indicates the total of *commits* that the repository received, of which C_{Stdt} indicates how many were made by the students, and C_{Tch} how many were made by the teacher. The last two columns present the average number of *commits* made by each of the students and the percentage of total commits that were made by the students.

TABLE III
PARTICIPATION IN THE SURVEY

Semester	Total students	Replies received	% answer
2017-2	28	2	7%
2018-1	22	4	18%
2018-2	33	2	6%
2019-1	32	3	9%
2019-2	31	8	26%
2020-1	34	10	29%
2020-2	44	5	11%

We can expecte the total number of *commits* per student to vary depending on the total number of submissions requested; in most cases, *commits* per student varies between 12 and 20. However, given that the total number of submissions required from students has remained between 9 and 11, the 2017-2 and 2018-2 semesters draws attention for having an average of *commits* per student close to half of what was observed in other semesters, and the 2020-1 semester for being close to double.

Regarding the first two cases, when explaining the way of working in *Git*, students were invited to make several *commits* in the development process of each of their deliveries, as would be done with a real development project. Students, however, tend to make *commits* only, since they see no motivation to do so throughout development. This is why, in the most elaborate project deliveries, completing more than five *commits* has been included as a criteria for grading; this can be shown to have measurably increased the total activity per student.

Regarding 2020-1, its high value is due to the fact that the course was taken by two students who already had experience using VCSs: One of them made 194 commits and the other 152, strongly raising the group average.

Regarding the students' appreciation on the usefulness of the knowledge acquired with the use of *Git*, although the experience had already been successful from the subjective point of view of the teacher and in informal conversations with some students, to be certain in this regard, an anonymous survey was applied to former students who took the course between semesters 2017-2 and to 2020-2; it was not sent to the first group to use *Git* (2017-1) as their email addresses were not gathered. The survey was generated on the free online platform *QuestionPro*. Participation in the survey was sought by sending a single email to the 224 students who were part of one of the groups in question. The survey was kept open for completion for a week during the month of November 2020, and 30 of the students submitted their responses. The distribution of responses over the groups is presented in Table III.

The survey consists of two multiple-choice questions, one single-choice question, five Likert scale questions, and two open-ended text questions. They were presented to the students divided into three groups: 1. Before the course, questions focused on knowing from which base points they arrived at the course; 2. During the course, how was their relationship with *Git* throughout the semester, and 3. After the course, how do they feel that learning *Git* has turned out for them after taking the course.

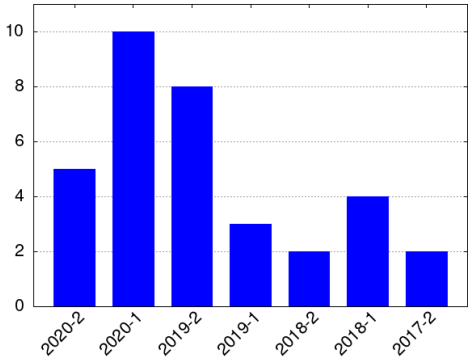

Fig. 3. During which semester did you take this course?

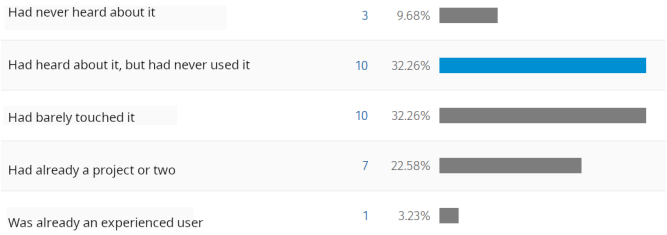

Fig. 4. Did you know anything about *Git* before taking this course?

The questions asked to the students were written in an informal and fun way, seeking to connect with them and achieve a higher response rate than what was estimated could be obtained with a traditional Likert scale, orienting the questions to a range of values of "very much in disagree" to "strongly agree".

A. Before the Course

Several students have taken this course more than once, which is why the first question is presented as multiple choice: During which semester did you take the Operating Systems course? (see Figure 3) The 30 attendees indicated 35 semestral participations, indicating that five among them were repaters.

We also asked the students how familiar they were with *Git* before taking the course; the results are presented in Figure 4.

It can be seen that, although a majority of students already knew something about *Git*, their familiarity was low: only 26% of the respondents had actively used it.

B. During the Course

The questions in this section were presented on a five-point Likert scale. The usual text of the Likert scale was replaced by semantic equivalents to facilitate the approach to the students.

As described above, basic handling of *Git* comes from only three practices. Is this enough of an introduction to know enough about the tool to use throughout the semester? Figure 5 shows that, indeed, the students considered it sufficient.

Having an explanation suffice does not necessarily mean the result of a tool results natural. Figure 6 shows that, while 16% considered it hard or very hard to use the tool, the majoritary opinion is still clearly positive.

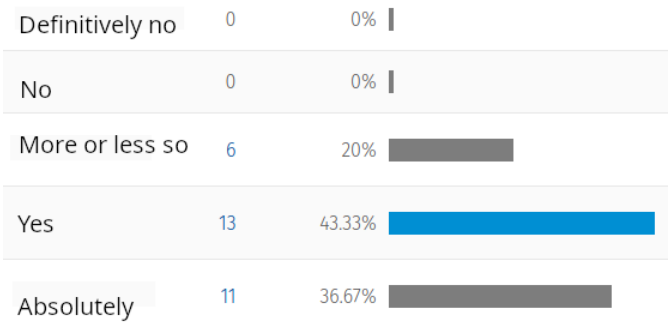

Fig. 5. Do you think the three practices on *Git*, plus the in-class explanations, were sufficient?

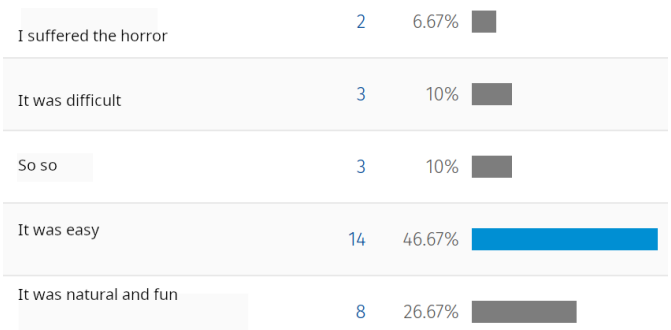

Fig. 6. Do you remember how hard it was to deliver your assignments and coordinate with your team partners?

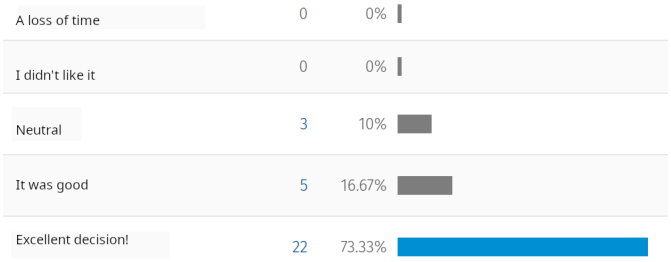

Fig. 7. How did using *Git* seem to you, in contrast with using a traditional LMS such as Moodle?

How does the use of *Git* answer to the original motivation of presenting an alternative to Moodle or other LMSs? Figure 7 shows the students resulted, in general, very satisfied with the change.

Using *Git* is only a part (albeit a fundamental one) of the experience. This experience was presented following the usual terminology and interaction found in the *GitHub* platform. All of the students but one indicated they were able to understand *GitHub*'s interaction model, and most of hem see value in its future use, as illustrated in Figure 8.

The last question in this section is open text, and it asks students to mention use cases or characteristics they would have wished to have covered. While some do mention characteristics that were not considered, such as issues, *stashes*, conflict resolution while *merging* changesets or others, the general feeling is of satisfaction with the level reached throughout the coursework.

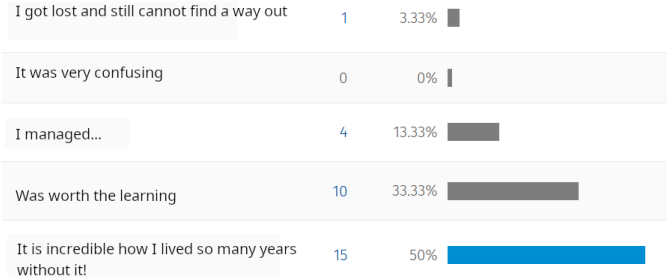

Fig. 8. It's not only *Git*: We used this system through the interaction with the GitHub web site. Did you think the workflow to be clear, with several branches, *clones* for managing students, *pull requests*, *pushes* and other actions?

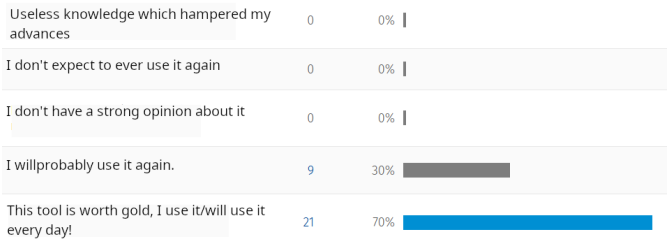

Fig. 9. Do you feel that learning about *Git* usage better enables you for your professional life?

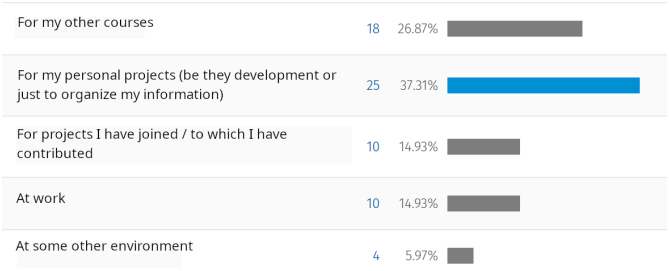

Fig. 10. Have you used *Git* again? Where?

C. After the Course

The teaching-learning process, however, must be evaluated against the long-term results for the students: How useful was the acquired knowledge to them? For this, a last question presented with a Likert scheme allows us to reach the closing of this tool adoption report with broad optimism: As Figure 9 shows, all students see value in it for their lives professional, 70% of them stating that *is/will be a daily-use tool*.

In the time elapsed between taking the course and when the survey was presented, have the students used *Git*? If so, where? This question *does not follow* a Likert-style scheme, but rather presents five possibilities, allowing for multiple choice. As 10 shows, 37% of the participants have used *Git* for their own (personal) projects, and 27% for other courses in their curricula.

Closing the survey, a final open question is given asking for additional comments. The feeling is overwhelmingly positive, with only three students mentioning the difficulty that the tool presented to them at the time; I'm taking the allowance to quote one of them:

> Ha ha, it's funny because I hated it during the course, because I didn't really understand much; however, once I got the hang of it, I became immensely fond of it and now we are inseparable; I'm not yet a primary, but I can defend myself. In fact, in my current job it was not used and I was the one who proposed it, and I practically trained my colleagues to learn how to use it, so in short, I am glad to have had an approach before where I could make a mistake and was explained where the issues I faced were.

These words sum up perfectly the overall intention of the teacher when presenting the use of *Git*: To provide an intellectual challenge that makes the students better suited for their professional life.

D. Threats to Validity

The author acknowledges that the experience report here presented is merely and precisely that: a single experience. In reading the text, it can be seen that the adoption of the interaction model does not derive from a rigorous approach to improving educational practices, but from an informal reflection. The approach was not made *in the dark*; particularly the architecture proposed by *GitHub Classrooms* [21] was considered, and fellow teachers were consulted looking for the best ideas and practices, but a full review of related works detailed in Section II was a consequence, rather than preparation, of the work done.

Although the total number of students who participated in the courses that used *Git* is not low (224 by the time this work was written), the fact that this work was not the result of planning from its onset caused the total number of students who responded to the survey to be barely 30. – and we cannot rule out that the application of the survey with up to three years of delay compared to the initial completion has introduced a bias in its results.

V. Conclusion

After an experience with eight semesters reported in this work, we believe it can be stated this experience has resulted clearly successful. Using *Git* has not been, as it has already been said, natural and clear for students of any discipline, but at least for intermediate and advanced courses of Computer Engineering, we invite teachers to adopt and improve upon the approaches here reported.

A. Future Work

In important part of the importance of the adoption of DVCSs lies in the collaboration between developers; particularly, DVCSs have been instrumental in facilitating collaboration in projects with high geographic dispersion. We consider it could be beneficial to add practices requiring two or more students to collaborate in the development of a given solution, each with their user, and deliver a practice combining their work. Although this has been spontaneously observed in the case of teamwork, it is not a skill that every student has developed. A practice following these lines has not been developed as we consider it to be unnecessarily complex for the coursework, but it would undoubtely help better present *Git* as a collaboration tool.

Managing submissions requires a nontrivial time commitment on the part of the teacher, particularly in ensuring that student submissions are made following the required naming convention. The development of *hooks* (self-running validators) to simplify this for the teacher has been considered, but they have not been implemented due to the potential they have to confuse the student with unexpected error messages.

Acknowledgment

The author would like to thank Laboratorio de Investigación y Desarrollo en Software Libre (LIDSOL), Facultad de Ingeniería, UNAM, for their help and support for this project.

References

[1] Z. Navarrete-Cazales and P. A. López Hernández, "La telesecundaria en México," *Perfiles Educativos*, vol. 44, no. 178, pp. 63–78, Oct. 2022, doi: 10.22201/iisue.24486167e.2022.178.60673.

[2] F. J. García-Peñalvo, "Estado actual de los sistemas e-learning," *Educ. Knowl. Soc.*, vol. 6, no. 2, pp. 1–7, Jan. 2022, doi: 10.14201/eks.18184.

[3] N. A. Alias and A. M. Zainuddin, "Innovation for better teaching and learning: Adopting the learning management system," *Malaysian Online J. Instructional Technol.*, vol. 2, no. 2, pp. 27–40, 2005. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d5f63457fb7c53eb83bd2149d860753c3ebca662

[4] M. J. Rochkind, "The source code control system," *IEEE Trans. Softw. Eng.*, vol. SE-1, no. 4, pp. 364–370, Dec. 1975. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6312866

[5] W. F. Tichy, "RCS—A system for version control," *Softw., Pract. Exper.*, vol. 15, no. 7, pp. 637–654, 1985, doi: 10.1002/spe.4380150703.

[6] B. Berliner, "CVS II: Parallelizing software development," in *Proc. USENIX Winter Tech. Conf.*, vol. 341, 1990, p. 352. [Online]. Available: https://www.tiffe.de/Robotron/PDP-VAX/rtVAX300/NetBSD6.0/usr/src/external/gpl2/xcvs/dist/doc/cvs-paper.pdf

[7] K. L. Reid and G. V. Wilson, "Learning by doing: Introducing version control as a way to manage student assignments," in *Proc. 36th SIGCSE Tech. Symp. Comput. Sci. Educ.* New York, NY, USA: ACM, Feb. 2005, vol. 37, no. 1, pp. 272–276. [Online]. Available: https://dl.acm.org/citation.cfm?id=1047441

[8] S. Williams, *Free as in Freedom: Richard Stallman's Crusade for Free Software*. Sebastopol, CA, USA: O'Reilly Media, 2011.

[9] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick, *Version Control With Subversion: Next Generation Open Source Version Control*. Sebastopol, CA, USA: O'Reilly Media, 2008.

[10] J. Brockmeier. *Counting Contributions: Who Wrote Linux 3.2*. Accessed: Jun. 14, 2019. [Online]. Available: https://www.linux.com/learn/counting-contributions-who-wrote-linux-32

[11] V. Henson and J. Garzik, "Bitkeeper for kernel developers," in *Proc. Ottawa Linux Symp.*, 2002, pp. 197–212. [Online]. Available: http://courses.cs.vt.edu/cs5204/fall05-gback/papers/ols2002_proceedings.pdf#page=197

[12] J. Barr. (2005). *Bitkeeper and Linux: The End of the Road*. [Online]. Available: https://www.linux.com/news/bitkeeper-and-linux-end-road

[13] B. de Alwis and J. Sillito, "Why are software projects moving from centralized to decentralized version control systems?" in *Proc. ICSE Workshop Cooperat. Human Aspects Softw. Eng.*, May 2009, pp. 36–39.

[14] K. Finley. (Jun. 2011). *GitHub Has Surpassed Sourceforge and Google Code in Popularity*. [Online]. Available: https://readwrite.com/github-has-passed-sourceforge/

[15] G. Wolf, "De Moodle a Git: Experiencia con el uso de un sistema de control de versiones (SCV) para reemplazar a un sistema de administración de la enseñanza (LMS)," in *Prácticas Abiertas* (Educación y Cultura Libre), A. Miranda, Ed. Tlalnepantla, Mexico: UNAM's Facultad de Estudios Superiores Iztacala (FES Iztacala), 2019, pp. 92–108. [Online]. Available: http://ru.iiec.unam.mx/4574/

[16] B. J. Cornelius, M. Munro, and D. J. Robson, "An approach to software maintenance education," *Softw. Eng. J.*, vol. 4, no. 4, pp. 233–236, 1989.

[17] L. Glassy, "Using version control to observe student software development processes," *J. Comput. Sci. Colleges*, vol. 21, no. 3, pp. 99–106, 2006. [Online]. Available: https://dl.acm.org/citation.cfm?id=1089195

[18] I. Milentijevic, V. Ciric, and O. Vojinovic, "Version control in project-based learning," *Comput. Educ.*, vol. 50, no. 4, pp. 1331–1338, May 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131506001977

[19] R. Glassey, "Adopting Git/GitHub within teaching: A survey of tool support," in *Proc. ACM Conf. Global Comput. Educ.*, May 2019, pp. 143–149. [Online]. Available: https://dl.acm.org/citation.cfm?id=3309518

[20] M. Pérez-Mateo Subirà and M. Guitert Catasús. (2011). *Aprender Y Enseñar En Línea*. [Online]. Available: http://cv.uoc.edu/annotation/e5274644a40912f5e2fbad5191bd9123/564161/PID_00173067/modul_1.html

[21] GitHub. (2019). *GitHub Classroom*. [Online]. Available: https://github.com/education/classroom/

Gunnar Wolf (Graduate Student Member, IEEE) received the bachelor's degree in software engineering from Secretaría de Educación Pública in 2011 and the master's degree in information security and technologies from Instituto Politécnico Nacional in 2018. He is currently pursuing the Ph.D. degree in computer science and engineering with Universidad Nacional Autónoma de México (UNAM). He is also an Academic Technical Staff with the Economics Research Institute (IIEc), UNAM, where he is also a Teacher with Facultad de Ingeniería (FI). He edited a freely redistributable textbook teaching operating systems in Spanish (https://sistop.org/) with FI, IIEc, UNAM, in 2015.