Extended version of the paper "Traceability by design: design of an interactive system to improve the automatic generation of Git traces during a learning activity"

Mika Pons, Jean-Michel Bruel, Jean-Baptiste Raclet, Franck Silvestre

Traceability *by design*: design of an interactive system to improve the automatic generation of Git traces during a learning activity

Mika Pons [1], Jean-Michel Bruel [2], Jean-Baptiste Raclet [3], and Franck Silvestre [1]

[1] IRIT, Université Toulouse Capitole, France
[2] IRIT, Université Toulouse 2 Jean Jaurès, France
[3] IRIT, Université Toulouse 3 Paul Sabatier, France

Abstract. *Learning Analytics* (LA) is collecting and analyzing traces of learners' activities in order to understand and improve learning. This paper focuses on traces generated using the version control system Git. Existing works on the topic have limitations regarding the quality of the traces they analyze: (1) the quantity and content are not always sufficient for in-depth analysis of student behavior, (2) their limited reliability can lead to a loss of exploitable data, and (3) the method of generating these traces is not generic. We propose a new interactive system based on Git and the observation of file modifications to generate automatically reliable and rich traces. This interactive system will soon be experimented with in an ecological context and is intended for diversified teaching contexts.

Keywords: Learning analytics, Git, Traces, Interactive system.

1 Introduction

The first conference on the field of *Learning Analytics* defines it as "the measurement, collection, analysis and communication of data about learners and their contexts, with the aim of understanding and optimizing the learning and the environments in which it occurs" [1].

If the use of version control tools, like Git, is widespread in the software industry, it is also used in an educational context, mainly in computer science training [7, 6]. Integrating such tools in teaching not only supports the professionalization of students but also facilitates collaborative work within a group of students. Finally, they constitute a simple means of obtaining traces of the activity of the students thanks to the actions of *commits* carried out during the progress of the activities to version the work. More particularly, Git works using a decentralized architecture. In fact, it allows to work with local copies and thus to synchronize with the main copy. This aspect is by nature very facilitating in the context of the teacher who must collect their students' work. Also, Git can store in an optimized way the modifications made by the students, called *diff*.

At the crossroads of *Learning Analytics* and version management software, we see the emergence of work based on the analysis of traces produced using version management software in computer learning [12]. Some works exploit *Process Mining* techniques for trace analysis by extracting metrics related to students' learning behavior [8, 3].

However, these studies have limitations concerning the traces generated by version control software (VCS) such as Git. Indeed, the content of these traces is not sufficient for certain analyses requiring more precise temporal information to be known. For example, we can't make an analysis on the duration of the resolution of exercises because there is not enough information available for that. Between the beginning of a work session and the resolution of an exercise, the time actually spent working is unknown. Also, these traces may lack reliability when their generation is based on a declarative and open process by the student. Finally, these traces are generated with methods that are difficult to apply in fields that do not fall within the field of computer science.

To overcome these limitations, we wonder how to design an interactive Git trace generation system for students, and more specifically:

RQ1 : How to increase the quantity and content of Git traces using an interactive system?
RQ2 : How to increase the reliability of Git traces using an interactive system?
RQ3 : How can an interactive system make the process of generating Git traces usable in disciplines other than computer science?

First, we will provide an overview of works that use Git traces to improve teaching in section 2, while highlighting their limitations. Secondly, in section 3, we will present the interactive system LAWG designed to answer the issues raised by our research questions. Finally, we'll explain how the features of LAWG address this in the section 4.

2 State of the art

Several papers [4, 7] have experimented with integrating VCS into computer science learning and show the benefits. Rocco et Lloyd [13] and Laadan et al. [6] show more precisely the interest of decentralized VCS, such as Mercurial and Git. In general, the integration of VCS into practical sessions gives the teacher the ability to centralize the distribution of course material for all students and to easily monitor and collect their work[2]. For the students, it introduces them to computer science industry standards[5].

In the field of *Process Mining* (PM), several works [3, 9, 15] focus on analyzing Git traces generated in the context of software development courses. In 2021, Macak et al. [8] proposed a method to analyze Git traces of student projects using PM. They are interested in 13 activities corresponding to the characteristics of a *commit*. More specifically, is it a *commit* containing essentially additions or deletions of lines, a big or a small *commit*, a *commit* related to branch management (new branch, *merge*, *pull request*), or a *commit* to create or modify

test files. This article shows it is possible to identify student learning behaviors using PM technique. However, only the information provided by Git in a *commit* is considered for analysis. Git traces do not contain contextual information about a student's progress in a learning activity, such as solving a well-identified question. There is no temporal information concerning the dates of the start of resolution activity for a question and therefore does not address (RQ1). Finally, this method of analysis does not deal with the issues of reliability and genericity of the generation of traces, as referred to in (RQ2, RQ3).

In 2018, Silvestre et Raclet [16] developed a learning protocol based on code review and test-guided development. In practice, students follow a worksheet and must mark the resolution of a question by performing *commits* on their associated Git repository. Corrections and code reviews are planned to consider the group's progress. To help orchestrate its different phases, a dashboard, G4S [11], is used. Based on the *commits* made and the milestones (reviews, corrections) filled in by the teacher, this dashboard provides a number of indicators, such as the progress of students within their group. The dashboard also makes it possible to export the traces enhanced by the context of the resolution of the questions.

We analyzed these traces with PM techniques [10] to extract indicators of students' learning behavior. Unfortunately, the results were limited by the lack of information contained in these traces. Indeed, it was impossible for us to precisely identify the beginning of an activity because the *commit* only gave the activity completion concerning a question, therefore not addressing (RQ1). In addition, the generation of traces identifies the completed exercise from a predefined message associated with the *commit*. This declarative and open process (the student can enter the message they want for the commit) has led to traces that are sometimes unusable due to errors in the message or omissions by the students, which raises the issue of the reliability of the traces mentioned in (RQ2). Also, the approach of manual generation of traces by students with Git is a problem in terms of genericity and applicability of the approach in other contexts (RQ3).

A 2022 study [14] explores an alternative way to automatically generate and visualize student activity traces during practical sessions. The data visualization platform EnCourse presents raw indicators directly related to the *commits* made by the students, such as the time spent on the project, the frequency of *commits*, or the number of lines added or removed. It also provides interpreted indicators such as the detection of cases of "academic dishonesty" or alerts when students are late. To automate the generation of the traces, Rodriguez-Rivera et al. have chosen to integrate the commit commands into a Makefile. Thus, each compilation makes a commit when the Makefile is executed.

We see three limitations regarding the traces generated via this operation. First, a compilation is not necessarily a very frequent event, and the tracing grain is, therefore, a bit coarse (RQ1). Secondly, using a Makefile is a constraint that makes it difficult to generalize to other fields than computer science (RQ3). Indeed, even in computer science, compilation is an action specific to program-

ming. A software engineering course where students aim is, for example, to write a software design documentation will not require compiling, and embedding a Makefile will be impossible. Finally, to calculate the time spent in activity, the platform counts, from the *commits*, fixed activity windows whose duration is chosen arbitrarily. However, there is nothing to fix *a priori* such a duration of activity for the student.

In conclusion of this state of the art, to our knowledge, there is no work on an interactive system making it possible to generate traces that lift the limits of content, reliability, and genericity we have just identified. We sought a compromise between G4S and EnCourse. Therefore, we have designed an interactive system to automatically generate traces that consider the need to have contextual information on students' progress in real-time while maximizing reliability and genericity.

3 The interactive LAWG system for the automatic generation of Git activity logs

In the context of solving worksheets given during practical sessions, each exercise can consist of several *questions*. We define the notion of *work session* as being a period of activity of a student, during which they manipulate resources (programming files, texts, images, ...), which constitute their *workspace*, to solve questions. A student can work outside supervised practical sessions, so a work session is not limited to practical sessions only.

A student's workspace exists locally on their machine and remotely. The remote space allows the teacher to access the student's work and to synchronize their local space if they need to work on several machines or with other students. When a student is done working on a question, they commit their changes, then move on to the next question. In our case, the workspace is managed with the VCS Git[4]. To commit its changes, it performs a *commit*, which saves locally all changes made to files since its last commit. The commit is always associated with a message entered by the student at the time of the commit (for example, `"Exercise 1 solved"`). To synchronize the remote workspace, their *repository*, with their local workspace, they execute the Git command `push`, and to synchronize their local space with the remote one, they execute the Git command `pull`. Finally, the sequence of commits on the workspace constitutes a *branch*. By default, there is a branch on which the student works, but they can choose to create a new one from this one. Branches evolve separately, so the student can switch branches to isolate specific types of changes (work on a feature, ...).

To answer the research questions posed in this paper, we designed an interactive system written in Python and open-source[5]. So that all students can

[4] https://git-scm.com/

[5] Available on GitHub at this address: https://github.com/git4school/LAWG.

```yaml
ssh_path: /home/user/.ssh/id_rsa
questions: []
groups: []
```

Listing 1: Structure of the file `settings.yml`

use it, it is available as an executable file for the three main operating systems: Windows, MacOS, and Linux.

By acting as an interface between the student and their Git repository, LAWG generates rich and reliable traces while abstracting Git for students.

3.1 Configuration

Configuration of the system is necessary from the teacher, during the preparation phase of the worksheet, and from the student, during the phase of working on the worksheet.

To prepare the worksheet, the teacher sets up a template workspace consisting of a required set of elements for the students to work on. In this workspace, they must integrate and configure LAWG (the executable files corresponding to the operating systems that their students are likely to use). The teacher must create the configuration file `settings.yml`, whose required fields are listed in the listing 1. We want the use of the system to be as simple and reliable as possible. For this, it is necessary to provide the list of questions the student must answer (the field `questions`). It is also at this step that the field `groups` can be filled in. Once done, the teacher can publish the worksheet and the template repository.

It's the student's turn to create their workspace from the template repository and complete the configuration. There remains only the field `ssh_path` in which they must indicate the path to the SSH key registered in the platform that hosts the remote Git repository. Student identification is one of the main issues with designing a dashboard like G4S. To do this, the students were asked to fill in the `README` file of their project with their names and group. The `README` could contain other information like instructions, so the dashboard had to extract the identity using a *parser*. In fact, the success of the identification of the student was strongly dependent on them, who had to (1) not forget to fill in the `README`, (2) fill in the `README` correctly, and (3) not modify the `README` later so the parsing can no longer be done correctly. To make this process more reliable, on the first start, LAWG asks the student to enter their name and group and generates the `IDENTITY.json` file (see listing 2 for a example). The dashboard will then read this file to identify each student.

Our interactive system offers three main functionalities that we explain in the following sections:

– Supervise work sessions, saving all resources at the end of each session;

```
{
    "first_name": "John",
    "last_name": "Doe",
    "group": "TPA12"
}
```

Listing 2: Structure of the file `IDENTITY.json`

– Observe all changes made to files in the workspace to save them in real-time;
– Provide a command-line interface for students to mark a question as solved.

3.2 Supervision of work sessions

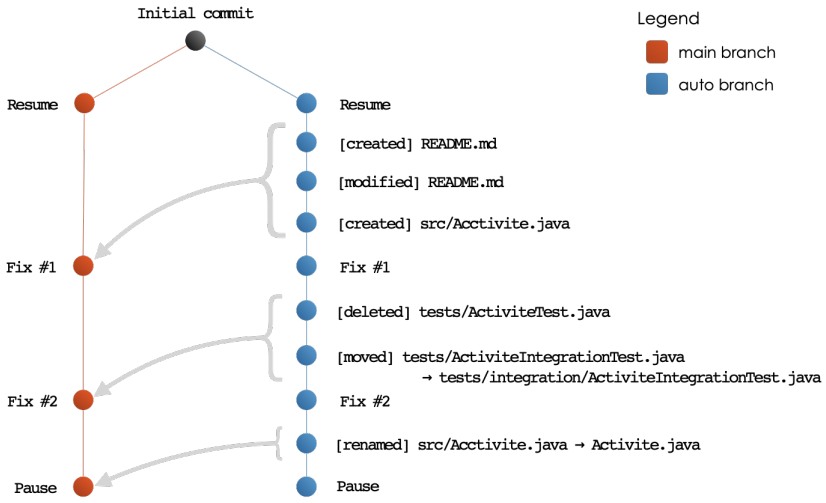


Fig. 1: Example of the use of the interactive system when carrying out practical exercises

When starting a work session, the student launches the interactive system. To trace this, the system creates a commit named `"Resume"` in the current branch and the `auto` branch (see figure 1). If the `auto` branch does not exist at that time, it is created from the current branch. The purpose of this branch will be explained in the section 3.3.

At the end of a work session, marked by the use of the `exit` command, all the changes in progress are saved under a `"Pause"` commit and sent to the remote workspace. Then all files in its local workspace are deleted. In this way, in the

eyes of the student, their workspace is empty. Only the system executable and the configuration file remain. The student must launch the interactive system to restore their workspace. The system can thus automatically trace the start and the end of a work session.

3.3 Automatic generation of activity traces

As soon as LAWG is launched, changes to workspace files are observed and give rise to system events. As shown in figure 1, we chose to use another branch, `auto`, to isolate the execution of automatic commits. This allows the student to have a simple current branch with few commits and the teacher to have a branch that tracks the complete activity of the student. Some commits are present in both branches. Each event causes a commit to be produced in the `auto` branch. The events observed are: *modification*, *creation*, *deletion*, *move* and *renaming* of a file.

For these automatic commits, we have defined a minimalist message form to facilitate further processing:

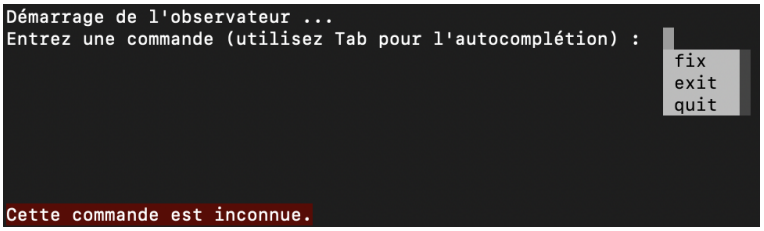
- `"[moved] <path_file> -> <path_new_folder>"`
- `"[renamed] <path_file> -> <new_name_file>"`
- `"[<event>] <path_file>"`

Thus, in the example in figure 1, we can see that the student starts their work session with the creation, then the modification of a `README` file, and creates a file `Acctivite.java` before marking question `#1` as solved. They then perform operations on test files, solve the `#2` question, realize that they have made a mistake on the name of a file, and correct the error. The student then closes their session, which has the effect of producing the commit `"Pause"`.

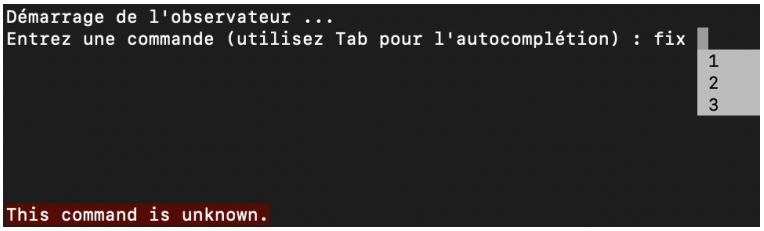
3.4 Exercise resolution: the *fix* command

Our interactive system also offers a command line interface, which notably provides the command `fix`. It allows the student to mark a question as solved, save the associated changes, and send them to the remote workspace. More precisely, all the changes since the last call to this same command, or since the launch of a working session, are validated in the form of a commit in the current branch and in the `auto` branch (whose message is of the form `"Fix <question>"`) and then sent to the remote workspace. In the example given in figure 1, the student used the `fix` command for the questions `#1` and `#2`, whose commits associated are visible in both branches, then the command `exit`, which produced the commit `"Pause"`.

The system suggests the available commands (cf. figure 2a) according to the entered characters. Also, the student cannot execute a command that does not exist or is incorrectly formed. When it recognizes the `fix` command, the interactive system displays the list of questions the student must answer (see figure 2b), previously filled in during configuration.



(a) Commands suggestion



(b) Questions suggestion

Fig. 2: Screenshots of LAWG's suggestions

The use of Github to manage practical work is very common in computer science. To orchestrate the resolution of questions on a worksheet, *issues* are sometimes used. It is then possible to close these issues, marking them as solved, directly in the commit message, by indicating the character `"#"` followed by the issue number to close. We have designed this interactive system so that it is possible to close these issues via the `fix` command. To do this, you need to define the questions to be resolved in the same way as the close issue key, i.e.: `"#1"` for the question and issue 1, ...

3.5 Limitations of the interactive system

This interactive system has limitations that must be taken into account before any integration.

The student still has to perform several actions manually. Indeed, they must register their public SSH key on Github when their workspace is hosted there, clone their repository for the first time, and use Git to do a pull to synchronize the workspace between two machines used possibly. These actions again imply a dependency on the student's mastery of Git. However, we are looking at ways to reduce the technical requirements.

While the LAWG retrieves recent changes to the remote repository via the *pull* command, it still cannot handle conflicts automatically. This can cause problems when several students are working on the same project.

When the students have to use several branches, the use of LAWG is not recommended at the moment. It does not handle branch switching correctly. If

the content of the workspace changes between two branches, the system will save all files different from the starting branch.

There is always a dependence on a declarative side on the part of the student, that is, the student must explicitly state the questions they solve bit by bit. There is no objective and automatic evaluation of the student's work.

4 What are the benefits of using the interactive system?

This section explains how the functionality of LAWG can address the three research questions set out in section 1.

4.1 RQ1 - Content

The generation of logs for each file modification (cf. section 3.3) makes it possible to trace a student's activity during a work session finely. Indeed, this finer grain of trace allows having the first sign of activity concerning a question *n* after the resolution of a question *n-1*. Thus, we can approach more precisely the real duration of activity of the student. To test the system, we generated traces of a student using the system to answer questions on an worksheet from a real course[6]. The figure 3 shows an extract of the generated traces. This extract shows only a subset of the data collected through Git. In this example, all lines in red are traces generated by the system. We can see that the volume of traces is significantly increased and covers a more extensive range of time than without using the system (i.e. only the commits named `"Fix ..."`).

Through the generation of a trace for each file modification, it is possible to identify information about students' behavior via the names of these files. A concrete example of this, in the case of the figure 3, is the identification of the good following of the Test-Driven-Development (TDD). We can see in lines 62 and 61 that the student integrates the test `ArticleTest`, then modifies the associated class `Article` thanks to the naming convention of the tests. This is, therefore, representative of TDD. On the other hand, we can see that the student modifies the classes `IndexController` on line 67 before the associated test `IndexControllerTest` on line 66. Here we see that the TDD is not followed, and a simple comparison of the file names would make it possible to automate this identification from these traces.

In addition, the management of work sessions (see section 3.2) makes integrating the beginning and end of these sessions in the logs possible. This can be observed in the figure 3 at the lines 44 and 74 with the commits `"Resume"` and `"Pause"`.

Also, the associated mechanism, which empties and restores the workspace, allows us to ensure the systematic use of the interactive system for each work session and, therefore, the completeness of the traces.

[6]The anonymized dataset is available on OSF at this address: https://osf.io/t3wqr/?view_only=69907570f39046edba382a5e855ed26a.

44	fbf16ec	Mon, 10 Apr 2023 18:28:55 +0200	Pause
45	ea3f9d1	Mon, 10 Apr 2023 18:26:46 +0200	Fix #2b
46	19c7009	Mon, 10 Apr 2023 18:26:06 +0200	[modified] src/test/java/doremi/doremi/BandTest.java
47	6890448	Mon, 10 Apr 2023 18:23:36 +0200	[modified] src/main/java/doremi/domain/Band.java
48	89a6b70	Mon, 10 Apr 2023 18:23:31 +0200	[modified] src/main/java/doremi/domain/Band.java
49	1157e96	Mon, 10 Apr 2023 18:23:15 +0200	[modified] src/test/java/doremi/doremi/BandTest.java
50	fa4054a	Mon, 10 Apr 2023 18:23:13 +0200	[created] src/test/java/doremi/doremi/BandTest.java
51	fa0c6fa	Mon, 10 Apr 2023 18:22:39 +0200	[modified] src/main/java/doremi/domain/Album.java
52	ce3c3bd	Mon, 10 Apr 2023 18:22:23 +0200	[modified] src/main/java/doremi/domain/Album.java
53	1c5090f	Mon, 10 Apr 2023 18:21:47 +0200	[modified] src/test/java/doremi/AlbumTest.java
54	2312a58	Mon, 10 Apr 2023 18:21:34 +0200	[modified] src/test/java/doremi/AlbumTest.java
55	533580e	Mon, 10 Apr 2023 18:21:33 +0200	[created] src/test/java/doremi/AlbumTest.java
56	ce3dd1d	Mon, 10 Apr 2023 18:19:26 +0200	Fix #2a
57	be88b27	Mon, 10 Apr 2023 18:19:11 +0200	[modified] src/main/java/doremi/domain/Article.java
58	6e9899f	Mon, 10 Apr 2023 18:19:07 +0200	[modified] src/main/java/doremi/domain/Article.java
59	6b853e4	Mon, 10 Apr 2023 18:18:21 +0200	[modified] src/main/java/doremi/domain/Article.java
60	6da0f26	Mon, 10 Apr 2023 18:17:59 +0200	[modified] src/main/java/doremi/domain/Article.java
61	ecb0a7a	Mon, 10 Apr 2023 18:17:29 +0200	[modified] src/test/java/doremi/ArticleTest.java
62	151e5c2	Mon, 10 Apr 2023 18:17:20 +0200	[created] src/test/java/doremi/ArticleTest.java
63	c13b362	Mon, 10 Apr 2023 18:16:09 +0200	Fix #1
64	fe51014	Mon, 10 Apr 2023 18:15:52 +0200	[modified] src/main/java/doremi/controllers/IndexController.java
65	e42f65f	Mon, 10 Apr 2023 18:14:54 +0200	[modified] src/test/java/doremi/IndexControllerTest.java
66	81bfb45	Mon, 10 Apr 2023 18:14:47 +0200	[created] src/test/java/doremi/IndexControllerTest.java
67	55f182b	Mon, 10 Apr 2023 18:14:22 +0200	[modified] src/main/java/doremi/controllers/IndexController.java
68	9f3e898	Mon, 10 Apr 2023 18:14:05 +0200	[modified] src/main/java/doremi/controllers/IndexController.java
69	424e099	Mon, 10 Apr 2023 18:13:43 +0200	[modified] src/main/java/doremi/controllers/IndexController.java
70	dff1b1d	Mon, 10 Apr 2023 18:13:28 +0200	[created] src/main/java/doremi/controllers/IndexController.java
71	3191ec0	Mon, 10 Apr 2023 18:12:33 +0200	[modified] src/main/resources/templates/index.html
72	9c8158d	Mon, 10 Apr 2023 18:12:24 +0200	[modified] src/main/resources/templates/index.html
73	79bbdbd	Mon, 10 Apr 2023 18:12:17 +0200	[created] src/main/resources/templates/index.html
74	8a0b7c7	Mon, 10 Apr 2023 18:10:07 +0200	Resume

Fig. 3: Extract of the traces generated with the system

4.2 RQ2 - Reliability

With the question resolution interface (cf. section 3.4), we went from declarative and open trace generation to declarative but closed trace generation. Indeed, the command `fix` generates a trace whose message is normalized. Validating the command prevents the student from marking as solved a question that is not in the list given to the configuration. In this way, errors from the students are avoided. Thanks to its partial automation (cf. section 3.3), traces generation becomes semi-automatic and closed. Many of the traces generated by LAWG are no longer dependent on student declaration, which reduces the possibility of omissions.

4.3 RQ3 - Genericity

The interactive system makes it easier to integrate into non-computer science areas. It abstracts the use of Git for trace generation so that it can be used for an audience unfamiliar with Git. The triggering of the automatic generation of traces (see section 3.3) when simply modifying a file makes it usable in all areas that require working on a computer, especially when the expected contributions are text productions.

5 Conclusion

After identifying the limitations of tools that generate traces through Git repositories, we presented the interactive system we designed to overcome them, called LAWG. Our interactive system allows the automatic generation of traces each time the student modifies a file. It also provides a command allowing the student to validate the resolution of a question. We thus contribute to improving the reliability, content, and genericity of the production of activity traces. With the enrichment of these traces, we intend to apply process mining methods or to perform analyses of student behavior in relation to a target behavior in future experiments. This year, our interactive system will be tested in computer learning courses for this purpose. To prepare for these experiments, our main objective is to deal with the issue of collecting personal data. This is a crucial point, all the more so in the case of an interactive system that observes the events of the student's machine in real-time and which, therefore, could substantially impact the system's acceptability. We have established an approach that separates access to personal data from research work. Only teachers have access to personal data via the dashboard G4S and make this data available to researchers once anonymized (through a script we provide). To facilitate this and ensure compliance with GDPR by construction, we plan to integrate data anonymization into the export from G4S.

Bibliography

[1] LAK '11: Proceedings of the 1st International Conference on Learning Analytics and Knowledge (2011)

[2] Clifton, C., Kaczmarczyk, L.C., Mrozek, M.: Subverting the fundamentals sequence: using version control to enhance course management. ACM SIGCSE Bulletin (2007)
[3] Eskofier, B.M.: Exploration of process mining opportunities in educational software engineering-the gitlab analyser. In: Proc. of The 13th International Conference on Educational Data Mining (EDM 2020) (2020)
[4] Glassy, L.: Using version control to observe student software development processes. Journal of Computing Sciences in Colleges (2006)
[5] Kelleher, J.: Employing git in the classroom. In: 2014 World Congress on Computer Applications and Information Systems (WCCAIS), IEEE (2014)
[6] Laadan, O., Nieh, J., Viennot, N.: Teaching operating systems using virtual appliances and distributed version control. In: Proceedings of the 41st ACM technical symposium on Computer science education (2010)
[7] Lawrance, J., Jung, S., Wiseman, C.: Git on the cloud in the classroom. Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13 (2013)
[8] Macak, M., Kruzelova, D., Chren, S., Buhnova, B.: Using process mining for Git log analysis of projects in a software development course. Education and Information Technologies (2021)
[9] Mittal, M., Sureka, A.: Process mining software repositories from student projects in an undergraduate software engineering course. In: Companion proceedings of the 36th international conference on software engineering (2014)
[10] Pons, M., Bruel, J.M., Raclet, J.B., Silvestre, F.: Finding behavioral indicators from contextualized commits in software engineering courses with process mining. In: Frontiers In Software Engineering Education, Springer (2023), to be published
[11] Raclet, J.B., Silvestre, F.: Git4School: A dashboard for supporting teacher interventions in software engineering courses. In: European Conference on Technology Enhanced Learning, Springer (2020)
[12] Robles, G., González-Barahona, J.M.: Mining student repositories to gain learning analytics. an experience report. In: 2013 IEEE Global Engineering Education Conference (EDUCON) (2013)
[13] Rocco, D., Lloyd, W.: Distributed version control in the classroom. In: Proceedings of the 42nd ACM technical symposium on Computer science education (2011)
[14] Rodriguez-Rivera, G., Turkstra, J., Buckmaster, J., LeClainche, K., Montgomery, S., Reed, W., Sullivan, R., Lee, J.: Tracking large class projects in real-time using fine-grained source control. In: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (2022)
[15] Shynkarenko, V., Zhevaho, O.: Application of constructive modeling and process mining approaches to the study of source code development in software engineering courses. Journal of Communications Software and Systems (2021)
[16] Silvestre, F., Raclet, J.B.: Développement dirigé par les tests et revue de code par les pairs pour l'apprentissage de la programmation. In: Ludovia

CH: 1ère édition sur le thème" Émanciper l'école et la société avec le numérique?" (2018)