Traçabilité *by design* : conception d'un système interactif pour améliorer la génération automatique de traces Git pendant une activité d'apprentissage

Mika Pons[1][0000−0002−5844−8727], Jean-Michel Bruel[2][0000−0002−3653−0148], Jean-Baptiste Raclet[3][0000−0001−7357−912X] et Franck Silvestre[1][0000−0002−1134−8200]

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

Résumé. Les *Learning Analytics* (LA) s'intéressent à la collecte et l'analyse des traces d'activités des apprenants dans le but de comprendre et améliorer leurs apprentissages. Dans cet article, nous nous intéressons aux traces laissées lors de l'utilisation de l'outil de gestion de versions Git. Les travaux existants sur le sujet présentent des limites quant à la qualité des traces qu'ils analysent : (1) la quantité et le contenu n'est pas toujours suffisant, (2) leur fiabilité relative peut entraîner une perte de données exploitables, et (3) la méthode de génération de ces traces n'est pas simple d'accès ou facilement applicable selon le public visé. Nous proposons un nouveau système interactif fondé sur Git et l'observation des modifications de fichiers afin de générer automatiquement des traces fiables et riches. Ce système sera expérimenté en contexte écologique prochainement et s'adresse à des contextes d'enseignement diversifiés.

Mots-clés : Learning Analytics, Traces d'apprentissage, Git, Système interactif

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

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

1 Introduction

La première conférence sur le champ des *Learning Analytics* le définit comme "la mesure, la collecte, l'analyse et la communication de données sur les apprenants et leurs contextes, dans le but de comprendre et d'optimiser l'apprentissage et les environnements dans lesquels il se produit" [1].

Si l'utilisation d'outils de gestion de versions est très répandue dans l'industrie du logiciel, on en retrouve également une exploitation dans un contexte éducatif, essentiellement dans l'apprentissage de l'informatique [5,4]. L'intégration de tels outils dans les enseignements permet non seulement d'accompagner la professionnalisation des étudiants, mais aussi de faciliter le travail collaboratif au sein d'un groupe d'étudiants. Ils constituent enfin un moyen simple d'obtenir des traces de l'activité des étudiants grâce aux actions de *commits* réalisées au cours de la progression des activités afin de versionner le travail.

À la croisée des *Learning Analytics* et des logiciels de gestion de versions, on voit émerger des travaux fondés sur l'analyse de traces produites avec l'utilisation de logiciels de gestion de versions dans l'apprentissage de l'informatique [10]. Certains travaux exploitent des techniques de *Process Mining* pour l'analyse des traces en extrayant des métriques liées au comportement d'apprentissage des étudiants [6,2].

Cependant, ces travaux présentent un certain nombre de limites concernant les traces générées grâce aux logiciels de gestion de versions tel que Git. En effet, le contenu de ces traces n'est pas suffisant pour certaines analyses nécessitant de connaître des informations temporelles plus précises. Aussi, ces traces peuvent être d'une fiabilité relative quand leur génération repose sur un processus déclaratif et ouvert de l'étudiant. Enfin, ces traces sont générées avec des méthodes que l'on peut difficilement appliquer avec des étudiants inexpérimenté avec Git.

Cela nous amène à nous demander quel système interactif de génération de traces à destination des étudiants concevoir pour générer des traces Git permettant de lever ces limites, et plus précisément :

RQ1 : Comment augmenter la quantité et le contenu de traces Git à l'aide d'un système interactif ?

RQ2 : Comment augmenter la fiabilité des traces Git à l'aide d'un système interactif ?

RQ3 : Comment un système interactif peut-il simplifier le processus de génération des traces Git pour une audience qui n'est pas familière avec Git ?

Dans un premier temps, nous ferons un tour d'horizon des travaux qui utilisent des traces Git pour améliorer l'enseignement en section 2, tout en mettant en lumière leurs limites. Dans un deuxième temps, en section 3, nous présenterons le système interactif *LAWG* conçu pour répondre aux problématiques soulevées par nos questions de recherche. Enfin, nous expliquerons comment les fonctionnalités de *LAWG* y répondent dans la section 4.

2 État de l'art

Plusieurs articles [3,5] ont fait l'expérience d'intégrer un logiciel de gestion de versions dans l'apprentissage de l'informatique et en montrent les bénéfices. [11] et [4] montrent plus précisément l'intérêt des logiciels de gestion de versions décentralisés, comme Mercurial et Git.

Dans le domaine du *Process Mining* (PM), plusieurs travaux [2,7,13] s'intéressent à analyser les traces Git générées dans le contexte de cours de développement logiciel. En 2021, [6] proposent une méthode pour analyser des traces Git de projets d'étudiants à l'aide du PM. Ils s'intéressent à 13 activités correspondant à des caractéristiques d'un *commit*. Plus précisément, est regardé s'il s'agit d'un commit contenant des additions ou des suppressions de lignes, s'il s'agit d'un gros ou d'un petit *commit*, d'un *commit* lié à la gestion des branches (nouvelle branche, *merge*, *pull request*), ou encore s'il s'agit d'un *commit* pour créer ou modifier des fichiers de test. Cet article montre qu'il est possible d'identifier des comportements d'apprentissages des étudiants en utilisant la technique du PM. En revanche, seules les informations fournies par Git dans un *commit* sont considérées pour être analysées. Les traces Git ne contiennent pas d'informations contextuelles quant à la progression d'un étudiant dans une activité pédagogique, comme la résolution d'une question bien identifiée. Il n'y a pas d'informations temporelles concernant les dates de début d'activité de résolution d'une question et donc n'adresse pas (RQ1). Enfin, cette méthode d'analyse ne traite pas les questions de la fiabilité de la génération des traces et de sa simplicité d'accès, comme visé dans (RQ2, RQ3).

En 2018, [14] développent un protocole d'apprentissage fondée sur la revue de code et le développement guidé par les tests. En pratique, les étudiants suivent une feuille d'exercices et doivent marquer la résolution d'une question en effectuant des *commits* sur leur dépôt Git associé. Des corrections et des revues de codes sont planifiées en tenant compte de la progression du groupe. Pour aider à l'orchestration de ces différentes phases, un tableau de bord, *Git4School* [9], est utilisé. À partir des *commits* réalisés par les étudiants et des jalons (revues de code, corrections) renseignés par l'enseignant, ce tableau de bord permet de visualiser un certain nombre d'indicateurs quant à la progression des étudiants par rapport à leurs camarades. Il permet également d'exporter les traces enrichies par le contexte de résolution des questions.

Nous avons analysé ces traces avec des techniques de PM [8] pour extraire des indicateurs sur le comportement d'apprentissage des étudiants. Malheureusement, les résultats étaient limitées par le manque d'informations contenues dans ces traces. En effet, il ne nous était pas possible d'identifier précisément le début d'une activité car le *commit* ne donnait que l'achèvement de l'activité concernant une question, n'adressant donc pas (RQ1). De plus, l'exercice terminé est identifié grâce à un message prédéfini associé au *commit*. Ce processus déclaratif et ouvert (l'étudiant peut entrer le message qu'il souhaite pour le commit) a conduit à des traces parfois inutilisables à cause d'erreurs dans le message ou d'oublis des étudiants, ce qui soulève l'enjeu de la fiabilité des traces mentionné dans (RQ2). Aussi, l'approche de génération manuelle des traces par

les étudiants avec Git pose un problème de simplicité d'accès et d'applicabilité de l'approche dans des contextes autres (RQ3).

Une étude de 2022 [12] explore un moyen alternatif de générer automatiquement et de visualiser des traces d'activité des étudiants lors de séances de TP. La plateforme de visualisation des données EnCourse présente des indicateurs bruts directement liés aux *commits* faits par les étudiants, comme le temps passé sur le projet, la fréquence des *commits*, ou le nombre de lignes ajoutées ou supprimées. Elle fournit également des indicateurs interprétés comme la détection de cas de "malhonnêteté académique" ou des alertes lorsque les étudiants sont en retard. Pour automatiser la génération des traces, [12] ont fait le choix d'intégrer les commandes de commit dans un fichier Makefile. Ainsi, un commit est fait chaque fois que le Makefile est exécuté, à chaque compilation.

Nous voyons trois limites concernant les traces générées via ce fonctionnement. Premièrement, une compilation n'est pas un événement fréquent et le grain de traçage est donc un peu grossier (RQ1). Deuxièmement, l'usage d'un fichier Makefile est une contrainte qui rend difficile la généralisation à d'autres domaines que celui de l'informatique (RQ3). En effet, et ce même en informatique, la compilation est une action spécifique à la programmation. Un cours de génie logiciel où les étudiants ont pour objectif, par exemple, de rédiger un dossier de conception ne nécessitera pas de compilation, et l'intégration d'un Makefile sera impossible. Enfin, pour calculer le temps passé en activité, la plateforme compte, à partir des *commits*, des fenêtres d'activité fixes dont la durée est choisie arbitrairement. Or rien ne permet de fixer a priori une telle durée d'activité de l'étudiant.

En conclusion de cet état de l'art, à notre connaissance, il n'existe pas de travaux sur un système interactif permettant de générer des traces qui lèvent les limites de contenu, de fiabilité et de simplicité que nous venons d'identifier. Pour ce faire, nous avons cherché un compromis entre *Git4School* et EnCourse. Nous avons donc conçu un système interactif de génération automatique de traces qui prend en compte le besoin d'avoir des informations contextuelles sur la progression des étudiants en temps réel, tout en maximisant la fiabilité des traces et la simplicité d'accès de leur méthode de production.

3 Le système interactif *LAWG* pour la génération automatique de traces Git d'activité

Nous nous plaçons dans le contexte de résolution de feuilles d'exercices[4] données lors de séances de travaux pratiques (TP). Ces exercices peuvent être constitués de plusieurs *questions*. Nous définissons la notion de *session de travail* comme étant une période d'activité d'un étudiant, durant laquelle il manipule des ressources (fichiers de programmation, textes, images, ...), qui constituent son *espace de travail*, pour résoudre des questions. Un étudiant peut travailler en dehors

[4]Un exemple de feuille d'exercices est donné ici : https://gist.github.com/raclet/c05729a4ae78082365ed0a1f8226fe55

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

Listing 1 : Structure du fichier `settings.yml`

des séances de TP encadrées, une session de travail ne se limite donc pas aux seules séances de TP.

L'espace de travail d'un étudiant existe localement sur sa machine, et à distance. L'espace à distance permet à l'enseignant d'accéder au travail de l'étudiant, et à ce dernier de synchroniser son espace local s'il doit travailler sur plusieurs machines ou avec d'autres étudiants. Lorsqu'un étudiant a fini de travailler sur une question, il valide ses changements, puis passe à la question suivante. Dans notre cas, l'espace de travail est géré avec le logiciel de gestion de versions Git[1]. Pour valider ses changements, il effectue un *commit*, ce qui enregistre localement toutes les modifications faites aux fichiers depuis son dernier commit. L'enregistrement est toujours associé à un message entré par l'étudiant au moment du commit (par exemple, `Exercice 1 résolu`). Pour synchroniser l'espace de travail distant, son *dépôt*, avec son espace local, il exécute la commande Git *push*, et pour synchroniser son espace local avec celui à distance, il exécute la commande Git *pull*. Enfin, l'enchaînement des commits sur l'espace de travail constitue une *branche*. Il existe une branche sur laquelle l'étudiant évolue par défaut, mais il peut faire le choix d'en créer une nouvelle à partir de celle-ci. Les branches évoluent de façon distincte, de sorte que l'étudiant peut changer de branche pour isoler certains types de modifications (travail sur une fonctionnalité, ...).

Pour répondre aux questions de recherche posées dans ce papier, nous avons conçu un système interactif écrit en Python et open-source[5]. Pour que tous les étudiants puissent l'utiliser, il est disponible sous forme de fichier exécutable pour les trois principaux systèmes d'exploitation : Windows, MacOS et Linux.

En agissant comme une interface entre l'étudiant et son dépôt Git, *LAWG* génère des traces enrichies et fiables tout en abstrayant Git pour les étudiants.

3.1 Configuration

Une configuration minimale est nécessaire, de la part de l'enseignant dans la phase de conception du sujet de TP, et de la part des étudiants en phase d'exécution du sujet.

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

[5] Disponible sur Github à cette adresse : https://github.com/git4school/LAWG.

Pour préparer le sujet, l'enseignant met en place un espace de travail "template", composé d'un ensemble requis d'éléments permettant aux étudiants de travailler. Dans cet espace de travail, il doit intégrer et configurer *LAWG* (les fichiers exécutables correspondant aux systèmes d'exploitations que ses étudiants sont susceptibles d'utiliser). L'enseignant doit créer le fichier de configuration `settings.yml`, dont les champs requis sont listés dans l'extrait 1. Nous voulons que l'utilisation du système soit la plus simple et fiable possible. Pour cela, il est nécessaire de lui fournir la liste des questions auxquelles l'étudiant devra répondre (le champs `questions`). C'est également à cette étape que les champs `groups` et `repo_path` peuvent être renseigner. Une fois cela fait, l'enseignant peut publier le sujet et le dépôt "template".

C'est au tour de l'étudiant de créer son espace de travail à partir du dépôt "template" et de terminer la configuration. Il ne reste que le champs `ssh_path` dans lequel il faut indiquer le chemin vers la clé SSH enregistrée dans la plateforme qui héberge le dépôt Git distant. Une des principales problématiques avec la conception d'un tableau de bord, comme *Git4School*, est l'identification des étudiants. Pour ce faire, il était demandé aux étudiants de remplir le fichier README de leur projet avec leur nom et leur groupe. Le README pouvait contenir d'autres informations comme des instructions, et le tableau de bord devait donc extraire l'identité à l'aide d'un *parser*. De fait, le succès de l'identification de l'étudiant était fortement dépendant de celui-ci, qui devait (1) ne pas oublier de remplir le README, (2) remplir le README correctement, et (3) ne pas modifier le README ultérieurement de sorte que le *parsing* ne peut plus se faire correctement. Afin de fiabiliser ce processus, au premier démarrage, *LAWG* demande à l'étudiant d'entrer son nom et son groupe et génère le fichier IDENTITY.json. Ce fichier sera ensuite lu par *Git4School* pour identifier chaque étudiant.

Notre système interactif offre 3 grandes fonctionnalités que nous explicitons dans les sous-sections suivantes :

- Encadrer les sessions de travail, sauvegardant toutes les ressources à la fin de chaque session ;
- Observer toutes les modifications apportées aux fichiers de l'espace de travail pour les sauvegarder en temps réel ;
- Offrir une interface en ligne de commande aux étudiants pour leur permettre de marquer une question comme résolue.

3.2 Encadrement des sessions de travail

Le début et la fin d'une session de travail sont jalonnés par des commits nommés `"Resume"` et `"Pause"` dans la branche courante. Comme illustré dans la figure 1, nous avons fait le choix d'utiliser une autre branche, `auto`, pour isoler l'exécution des commits automatiques. Cette branche est créée à partir de la branche courante au lancement de *LAWG* si elle n'existe pas. Cela permet à

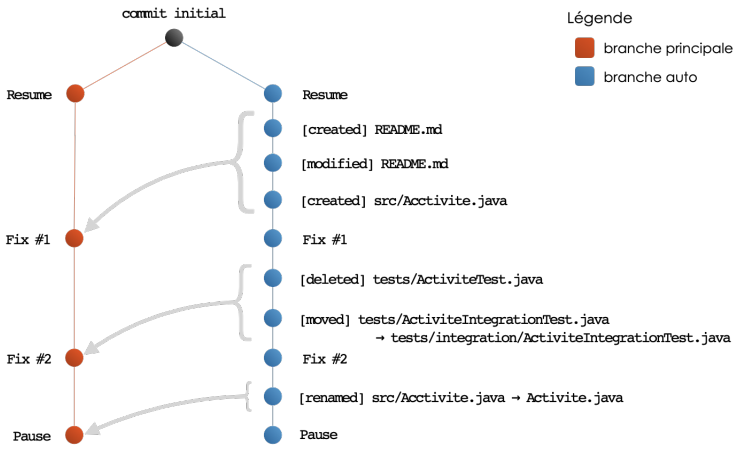


FIG. 1 : Exemple de l'utilisation de *LAWG* lors de la réalisation d'exercices de travaux pratiques

l'étudiant d'avoir une branche courante simple avec peu de commits, et à l'enseignant d'avoir une branche qui retrace l'activité complète de l'étudiant. Certain commits sont présents dans les deux branches.

Lors de la fin d'une session de travail, marquée par l'utilisation de la commande `exit`, tous les changements en cours sont sauvegardés sous un commit `"Pause"` et envoyés dans l'espace de travail distant. Ensuite, tous les fichiers de son espace de travail local sont supprimés. De cette façon, aux yeux de l'étudiant, son espace de travail est vide, il ne reste plus que l'exécutable du système et le fichier de configuration. L'étudiant doit lancer le système interactif pour restaurer son espace de travail. Le système peut ainsi tracer automatiquement le démarrage d'une nouvelle session de travail, représenté par un commit `"Resume"` dans la branche `auto`.

3.3 Génération automatique des traces d'activité

Dès le lancement de *LAWG*, les modifications des fichiers de l'espace de travail sont observées et donnent lieu à des événements système. Chaque événement entraîne la production d'un commit dans la branche `auto`. Les événements observés sont : la *modification*, la *création*, la *suppression*, le *déplacement* et le *renommage* d'un fichier.

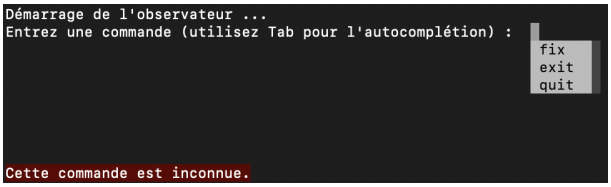
Pour ces commits automatiques, nous avons défini une forme de message minimaliste pour faciliter le traitement par la suite :

- `"[moved] <chemin_fichier> -> <chemin_nouveau_dossier>"`
- `"[renamed] <chemin_fichier> -> <nouveau_nom_fichier>"`
- `"[<evenement>] <chemin_fichier>"`

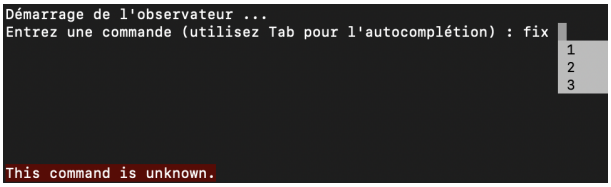
Ainsi, dans l'exemple de la figure 1, on peut voir que l'étudiant démarre sa session de travail avec la création, puis la modification d'un fichier README, et crée un fichier `Acctivite.java` avant de marquer la question `#1` comme résolue. Il fait ensuite des opérations sur des fichiers de tests, résout la question `#2`, puis se rend compte qu'il a fait une faute sur le nom d'un fichier et corrige l'erreur. L'étudiant ferme ensuite sa session, ce qui a pour effet de produire le commit `"Pause"`.

3.4 Interface de résolution d'un exercice

Notre système interactif propose également une interface en ligne de commande, qui met notamment à disposition la commande `fix`. Elle permet à l'étudiant de marquer une question comme résolue, de sauvegarder les changements associés et de les envoyer dans l'espace de travail distant. Plus précisément, tous les changements depuis le dernier appel à cette même commande, ou depuis le lancement d'une session de travail, sont validés sous forme d'un commit dans la branche courante et dans la branche `auto` (dont le message est de la forme `"Fix <question>"`) puis envoyés dans l'espace de travail distant. Dans l'exemple donné en figure 1, l'étudiant a utilisé la commande `fix` pour les questions `#1` et `#2`, dont les commits associés sont visibles dans les deux branches, puis la commande `exit`, qui a produit le commit `"Pause"`.



(a) Suggestion des commandes



(b) Suggestion des questions

FIG. 2 : Captures d'écran des suggestions de *LAWG*

Le système offre une suggestion des commandes disponibles (cf. figure 2a) en fonction des caractères déjà entrés et l'étudiant ne peut pas lancer une commande qui n'existe pas ou qui est mal formée. Lorsqu'il reconnaît la commande `fix`, le système interactif affiche la liste des questions auxquelles l'étudiant doit répondre (cf. figure 2b), renseignées auparavant lors de la configuration. Le système a

été conçu de manière à pouvoir spécifier et implanter des commandes selon le contexte d'enseignement facilement.

3.5 Limites du système interactif

LAWG a des limites qu'il faut prendre en compte avant toute intégration.

L'étudiant doit encore effectuer plusieurs actions manuellement. En effet, il doit enregistrer sa clé SSH publique sur Github lorsque son espace de travail y est hébergé, cloner son dépôt pour la première fois et utiliser Git pour faire un pull afin éventuellement de synchroniser l'espace de travail entre deux machines utilisées. La nécessité d'avoir recours à une clé SSH est un frein à lever pour lequel nous étudions des pistes simplificatrices car nous visons à terme un public non informaticien.

De plus, *LAWG* ne récupère pas les changements récents faits sur le dépôt distant via la commande *pull* car il ne pourrait pas gérer les éventuels conflits de manière automatique. Cela peut poser des problèmes lorsque plusieurs étudiants travaillent sur un même projet ou si un étudiant travaille sur plusieurs machines différentes.

Dans le cas où les étudiants sont amenés à utiliser plusieurs branches, l'utilisation de *LAWG* est pour le moment déconseillée. Il ne gère pas correctement le changement de branche. Si le contenu de l'espace de travail change entre deux branches, le système va sauvegarder tous les fichiers différents de la branche de départ.

Il subsiste une dépendance au déclaratif de la part de l'étudiant, c'est-à-dire que l'étudiant doit déclarer explicitement les questions qu'il résout petit à petit. Il n'y a pas d'évaluation objective et automatique du travail de l'étudiant.

4 Quelles sont les bénéfices de l'utilisation du système interactif ?

Dans cette section, nous expliquons comment les fonctionnalités de *LAWG* permettent de répondre aux trois questions de recherche énoncées dans la section 1.

4.1 RQ1 - Contenu

La génération des traces pour chaque modification de fichier (cf. sous-section 3.3) permet de tracer finement l'activité d'un étudiant pendant une session de travail. En effet, ce grain plus fin de trace permet d'avoir le premier signe d'activité concernant une question *n* après la résolution d'une question *n-1*. Ainsi, nous pouvons nous approcher plus précisément de la durée réelle d'activité de l'étudiant. En générant une trace pour chaque modification de fichier, il est possible d'identifier des informations sur le comportement des étudiants via les noms de ces fichiers. Un exemple concret, dans le cas de la figure 3, est l'identification du bon respect du Test-Driven-Development (TDD). Cette figure est un extrait

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 : Extrait des traces générées avec *LAWG*

des traces que nous avons générées pour tester le système[6]. Elles simulent un étudiant utilisant le système pour répondre à des questions sur une feuille de travail d'un cours réel. Chaque ligne en rouge correspond à une trace générée par *LAWG*. Nous pouvons voir que le volume des traces est significativement augmenté et couvre une plage de temps plus étendue que sans l'utilisation du système (seulement les commits nommés `"Fix ..."`). Nous pouvons voir aux lignes 62 et 61 que l'étudiant intègre le test `ArticleTest`, puis modifie la classe associée `Article` grâce à la convention de nommage des tests. Ceci est donc représentatif du TDD. D'autre part, nous pouvons voir que l'étudiant modifie les classes `IndexController` à la ligne 67 avant le test associé `IndexControllerTest` à la ligne 66. On voit ici que le TDD n'est pas respecté, et comparer les noms de fichiers permettrait d'automatiser cette identification à partir de ces traces.

De plus, la gestion des sessions de travail (cf. sous-section 3.2) permet d'intégrer le début et la fin de ces sessions dans les traces. Aussi, le mécanisme associé qui vide et restaure l'espace de travail, permet de nous assurer de l'utilisation

[6]Les données sont disponibles en accès libre sur https://osf.io/t3wqr/?view_only=69907570f39046edba382a5e855ed26a

systématique du système interactif pour chaque session de travail, et donc de l'exhaustivité des traces.

4.2 RQ2 - Fiabilité

Avec l'interface de résolution de question (cf. sous-section 3.4), nous sommes passés d'une génération de traces déclarative et ouverte à une génération de traces déclarative mais fermée. En effet, la commande `fix` génère une trace dont le message est normalisé. La validation de la commande empêche l'étudiant de marquer comme résolue une question qui n'est pas dans la liste donnée à la configuration. On évite de cette façon les erreurs de la part des étudiants. Grâce à son automatisation en partie (cf. sous-section 3.3), la génération de traces devient semi-automatique et fermée. Une grande partie des traces générées par *LAWG* n'est plus dépendante de la déclaration de la part de l'étudiant, ce qui réduit la possibilité d'oublis.

4.3 RQ3 - Simplicité d'accès

Le système interactif permet de s'intégrer plus facilement dans des domaines non informatiques. Il abstrait l'utilisation de Git pour la génération des traces, et peut donc être utilisé pour une audience avec un profil en informatique mais qui n'est pas familiarisé ave Git. Le déclenchement de la génération automatique des traces (cf. sous-section 3.3) lors de la simple modification d'un fichier est un pas en faveur de l'utilisation du système dans n'importe quel domaine nécessitant de travailler sur ordinateur, notamment quand les contributions attendues sont des productions de textes.

5 Conclusion

Après avoir identifié des limites des outils qui génèrent des traces à travers des dépôts Git, nous avons présenté le système interactif que nous avons conçu pour les lever, appelé *LAWG*. Notre système interactif permet la génération automatique de traces à chaque modification de fichier par l'étudiant. Il fournit également une commande permettant à l'étudiant de valider la résolution d'une question. Nous contribuons ainsi à abstraire la production de traces d'activité, et à améliorer leur fiabilité et leur contenu. Notre système interactif sera expérimenté cette année dans des cours d'apprentissage de l'informatique pour produire de nouvelles traces. Nous avons établi une approche qui sépare l'accès aux données personnelles du travail de recherche. Seuls les enseignants ont accès aux données personnelles via le tableau de bord *Git4School* et mettent ces données à la disposition des chercheurs une fois anonymisées (au moyen d'un script que nous fournissons). Pour faciliter cette démarche et assurer la conformité au RGPD par conception, nous prévoyons d'intégrer l'anonymisation des données dans l'exportation à partir de *Git4School*.

Références

1. LAK '11 : Proceedings of the 1st International Conference on Learning Analytics and Knowledge (2011)
2. 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)
3. Glassy, L. : Using version control to observe student software development processes. Journal of Computing Sciences in Colleges (2006)
4. 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)
5. 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)
6. 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)
7. 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)
8. 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 (2023)
9. Raclet, J.B., Silvestre, F. : Git4School : A dashboard for supporting teacher interventions in software engineering courses. In : European Conference on Technology Enhanced Learning (2020)
10. 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)
11. Rocco, D., Lloyd, W. : Distributed version control in the classroom. In : Proceedings of the 42nd ACM technical symposium on Computer science education (2011)
12. 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)
13. 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)
14. 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)