Mise en oeuvre d'approches pédagogiques fondées sur des pratiques de l'industrie du logiciel pour l'apprentissage de la programmation

Jean-Baptiste Raclet, Franck Silvestre, Mika Pons

Mise en œuvre d'approches pédagogiques fondées sur des pratiques de l'industrie du logiciel pour l'apprentissage de la programmation

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

[1] IRIT, Institut de Recherche en Informatique de Toulouse, France.
nom.prenom@irit.fr
[2] Université Paul Sabatier – Toulouse III, France.
[3] Université Toulouse Capitole – Toulouse I, France.

Résumé Cet article présente un nouveau protocole d'apprentissage de la programmation dans lequel les séquences de travail sont structurées en cycles : d'abord l'étudiant développe individuellement une partie d'un logiciel spécifiée par un ensemble exhaustif de tests automatisés fournis par l'enseignant. Ensuite, afin de favoriser les interactions entre pairs, une ou plusieurs phases de revue de code sont introduites. Ces phases de revue sont orchestrées avec la plate-forme elaastic.

L'orchestration du protocole est facilitée grâce à l'outil Git4School qui extrait des métriques d'apprentissage depuis les dépôts git des étudiants et les synthétise sous forme de graphiques permettant un suivi individualisé des étudiants.

Keywords: Ingénierie logiciel, conception guidée par les tests, revue de code, apprentissage de la programmation, évaluation par les pairs

1 Introduction

La formation de développeurs logiciel est un enjeu majeur comme l'indique la tension du marché de l'emploi. En 2017, La Direction de l'Animation de la Recherche, des Études et des Statistiques (DARES) rattachée au ministère français du travail rapporte un indicateur de tension de 30% concernant les ingénieurs en informatique [Bergeat, 2017], catégorie regroupant essentiellement *« les analystes programmeurs, chefs de projet et ingénieurs d'études ou de développement »* [Colin et al., 2015]. Dans le même temps, cet indicateur est de 4% pour l'ensemble du marché du travail.

En conséquence, une employabilité immédiate des jeunes diplômés est un défi qui consiste non seulement à former des étudiants à une discipline en terme de compétences techniques et méthodologiques mais aussi à transmettre les pratiques courantes de l'industrie du logiciel. Cela s'avère aussi essentiel pour la mise en œuvre de prestations de formation tout au long de la vie (FTLV) pour satisfaire les objectifs de mises-à-jour des connaissances ou reconversion professionnelle des participants.

Le SWEBOK [Bourque and Fairley, 2014], *Guide to the Software Engineering Body of Knowledge*, est un référentiel établi par l'IEEE qui structure les connaissances relatives au développement logiciel et propose une synthèse des pratiques, techniques et normes généralement acceptées dans l'ingénierie du logiciel. Il fait ainsi un inventaire méthodologique qui permet d'orienter le développeur. Dans le même esprit, le curriculum de l'ACM/IEEE en ingénierie logiciel [Sahami et al., 2013] est un ensemble de recommandations, établies par des professionnels conjointement avec des représentants du monde académique et régulièrement mis à jour, pour l'établissement de cursus d'apprentissage. Il identifie des corpus de connaissances, des dépendances entre eux et définit un degré d'importance pour chacun.

L'approche la plus courante dans l'enseignement supérieur est de considérer que la professionnalisation des pratiques s'effectue lors de projets longs, qui sont l'occasion d'expérimenter le travail en groupe, ou alors au cours d'immersion en entreprise lors de stages. Dans cet article, nous arguons que celles-ci peuvent être enseignées plus tôt en adaptant les pratiques pédagogiques utilisées. Pour cela, nous proposons en section 2 un nouveau protocole fondé sur le test et la confrontation de points de vue pour l'apprentissage de la programmation. Ce protocole inclut des outils et méthodes reconnues par les professionnels et mentionnées dans le SWEBOK ; il permet aussi de mieux couvrir les objectifs d'apprentissage définis dans le curriculum de l'ACM/IEEE. Ainsi, il autorise la mise en œuvre de nouveaux scénarios pédagogiques permettant aux élèves de consolider les connaissances et de se former aux pratiques industrielles de l'ingénierie du logiciel.

Dans le protocole proposé, le travail de l'étudiant comporte différentes tâches décomposées en plusieurs phases décrites ultérieurement dans cet article. Pour faciliter son orchestration, nous avons développé l'outil Git4School, présenté en section 3. Cet outil à destination des enseignants constitue un tableau de bord exposant différentes métriques d'apprentissage extraites des dépôts *git* contenant le travail individuel des étudiants permettant ainsi un suivi individualisé des apprenants.

Le protocole exposé ici est expérimenté depuis 2 ans dans le cadre d'enseignements introductifs et avancés dans le supérieur (en Licence professionnelles, en première et seconde année d'un Master spécialisé dans l'ingénierie logiciel) portant sur l'utilisation du *framework* Java de persistance des données JPA. Un retour sur ces expérimentations ainsi qu'un positionnement par rapport à des travaux connexes est effectué en section 4. Enfin, la conclusion présente les idées clés qui guideront nos travaux futurs.

2 Un protocole fondé sur le test et la revue par les pairs pour l'apprentissage de la programmation

Cette section décrit un nouveau protocole de travail à partir duquel des scénarios pédagogiques peuvent être établis. Le processus est gouverné par les principes suivants :

— Chaque étudiant travaille individuellement à développer une partie d'un logiciel dans lequel les fonctionnalités principales peuvent être réalisées avec des éléments de programmation ciblés.
— Pour chaque exercice, des consignes sont données de manière traditionnelle, avec une description en langage naturel.
— Les exercices demandant d'écrire du code viennent toujours avec un ensemble de tests automatisés qui doivent s'exécuter avec un verdict de succès pour considérer l'exercice comme effectué correctement.
— Lorsque l'étudiant a produit sa solution à un exercice et que tous les tests correspondants s'exécutent avec un verdict de succès, l'étudiant effectue un *commit* dans un dépôt *git* privé qui lui a été attribué.
— Chaque séquence de travail est structurée en une succession de cycles durant lesquels l'étudiant travaille d'abord individuellement, en autonomie (phase de travail guidé par les tests) puis s'en suivent des activités dans lesquelles les étudiants sont invités à partager leur solution, évaluer d'autres solutions et éventuellement ensuite adapter leur propre solution (phase de revue par les pairs).

Nous présentons maintenant en détails les phases mentionnées.

2.1 Phases de travail en solo guidée par les tests (TDD)

Dans l'ingénierie du logiciel, les tests représentent un dispositif incontournable pour vérifier et valider un système logiciel [Bourque and Fairley, 2014]. Le développement dirigé par les tests (TDD pour *Test Driven Development*) est une pratique d'ingénierie, formalisée initialement dans le cadre de la méthode XP, qui consiste à développer les fonctionnalités d'un logiciel dans une succession de cycles commençant par l'écriture d'un test, suivi de l'écriture du code faisant passer le test et terminant par l'amélioration du code principal (*refactoring*). Cette approche redéfinit complètement la place des tests en ingénierie logicielle : le TDD est une méthode d'analyse, de conception et de développement reposant sur l'écriture de tests automatisés tout au long du cycle de développement [Beck, 2003] et non plus uniquement à des fins de validation.

Le protocole que nous définissons requiert que chaque exercice soit équipé d'un ensemble de tests unitaires et de tests d'intégration. L'étudiant suit alors la philosophie du TDD et travaille par itérations successives sur son code jusqu'à ce que les tests s'exécutent avec un verdict de succès.

La validation des compétences et l'obtention de crédits ECTS étant individuels, les évaluations sommatives le sont aussi. Afin de conserver un alignement pédagogique tel que décrit dans [Biggs, 1996], il convient donc de proposer aux étudiant des activités individuelles. Les phases TDD durant lesquelles les étudiants travaillent seuls remplissent cette part du contrat pédagogique.

Le tableau 1 liste à travers l'étude d'un exemple les principales caractéristiques des tests fournis aux étudiants et leur traduction en terme de spécification sur ce que doit faire l'étudiant. On remarque que les tests peuvent être utilisés pour spécifier ce que le code principal doit faire (caractéristique 2) mais aussi comment il doit le faire (caractéristiques 1 et 3).

Id.	Caractéristique	Exemple

1	Un test ne compile que si les types de données manipulés dans le test sont créés correctement par l'étudiant dans le programme principal (idem pour les méthodes).	`// given: an enterprise with all`
		`// properties correctly set`
		`enterprise = new Enterprise();`
		`enterprise.setName("Company & Co");`
		`enterprise.setDescription("Comp desc");`

2	Les tests dits "fonctionnels" (qui testent le comportement attendu en mode boite noire) ne passent que si l'étudiant a codé les algorithmes qui permettent d'atteindre les états des objets spécifiés par les tests.	`// when we switch between, enterprise`
		`// 1 and 2 on project 1`
		`project.switchEnterprise(enterprise2);`
		`// and we save the project`
		`Project savedProject =`
		`enterpriseProjectService.save(project);`
		`// then the saved project is attached`
		`// to enterprise 2`
		`assertThat(savedProject.getEnterprise(),`
		`is(enterprise2));`
		`// and enterprise 1 has only one`
		`// associated project`
		`assertThat(enterprise1.getProjects().`
		`size(), is(1));`

3	Les tests dits "système" ou d'interactions permettent de forcer certains aspects de la solution : utilisation d'un composant précis ; collaboration entre objets, ...	`// when: trying to find the project by id`
		`enterpriseProjectService.`
		`findProjectById(anId);`
		`// then: the find operation is triggered`
		`// on the entity manager`
		`verify(entityManager).`
		`find(Project.class, anId);`

4	Les exécutions successives d'un test génèrent une succession de cycles essais-erreurs où les résultats des tests fournissent un feedback	La figure 1 présente un exemple de feedback obtenu suite à l'exécution d'un test en échec.

Table 1. Caractéristiques des tests

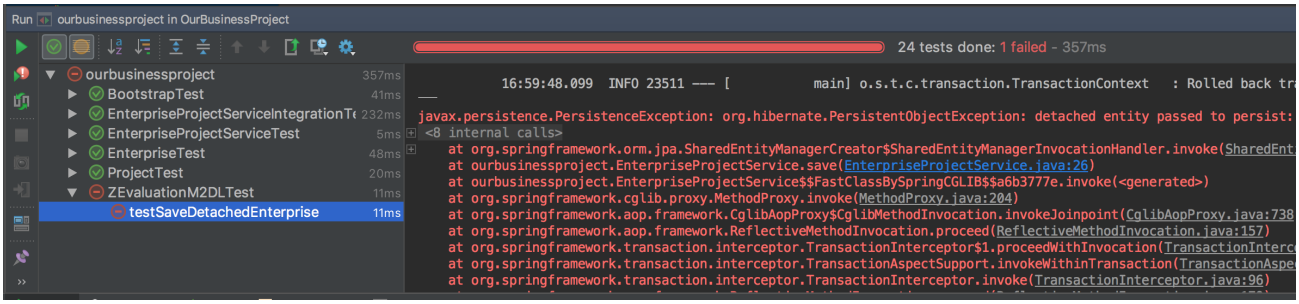

Figure 1. Un feedback obtenu suite à l'exécution d'un ensemble de tests

Un grand nombre d'objectifs d'apprentissage spécifiques à la programmation présentés dans [Sahami et al., 2013] sont couverts par les phases guidées par les tests ; le tableau 2 en mentionne quelques-uns. Néanmoins certains objectifs

d'apprentissage requièrent des interactions sociales comme cet attendu décrit dans [Sahami et al., 2013] : « Discuter des moyens de résoudre un problème avec différents algorithmes ayant chacun leurs propres propriétés ». Afin d'atteindre ces objectifs, le protocole inclut à la suite des phases TDD des phases de revue par les pairs que nous introduisons dans la partie suivante.

Id.	Objectif d'apprentissage	Phase
1	Analyser le comportement de programmes simples impliquant les éléments de base suivants : variables,expressions, affectations, E / S, structures de contrôle, fonctions, passage de paramètres et récursivité.	TDD
2	Concevoir, implémenter, tester et déboguer un programme qui utilise chacune des structures de programmation suivantes : calcul de base, E / S simples, structures conditionnelles et itératives, définition de fonctions et passage de paramètres.	
3	Écrire un programme utilisant des E / S de fichiers pour assurer la persistance entre plusieurs exécutions.	
4	Écrire des programmes qui utilisent chacune des structures de données suivantes : tableaux, enregistrements structures, chaînes, listes chaînées, piles, files d'attente, ensembles et cartes.	
5	Construire, exécuter et déboguer des programmes en utilisant les environnements de développement intégrés modernes	
6	Construire et déboguer des programmes en utilisant les bibliothèques standardisées disponibles pour un langage de programmation donné	
7	Discuter des moyens de résoudre un problème avec différents algorithmes ayant chacun leurs propres propriétés.	PR
8	Expliquer le comportement de programmes simples impliquant les éléments de base suivants : variables, expressions, affectations, E / S, structures de contrôle, fonctions, passage de paramètres et récursivité.	
9	Participer à une révision du code d'une petite équipe axée sur l'exactitude des composants.	
10	Analyser dans quelle mesure le code d'un autre programmeur respecte la documentation et les standards d'un style de programmation.	

Table 2. Extrait des objectifs d'apprentissage couverts par le protocole proposé

La phase TDD favorise un apprentissage où chaque étudiant avance à son rythme et, dans le meilleur des cas, un apprentissage auto-régulé. En effet, les étudiants travaillent sur des exercices en étant guidés par les *feedbacks* fournis par l'exécution des tests. Lors de nos expérimentations, cette approche a permis à certains étudiants d'appréhender et de comprendre seuls les notions abordées. En conséquence, l'enseignant dispose de davantage de temps à consacrer aux étudiants ayant des difficultés sur certains exercices.

2.2 Phases de revue de code par les pairs (PR)

A l'issue des phases de TDD, les étudiants arrivent avec une solution correcte vis-à-vis des tests fournis. L'objectif des phases de revue de code par les pairs (PR pour *peer review*) est de confronter les différentes solutions entre elles afin d'approfondir la compréhension par les étudiants des concepts sous-jacents aux exercices. Elles visent aussi à s'exercer aux activités sociales qui sont centrales dans des processus collaboratifs de développement ([Bourque and Fairley, 2014], chapitre 8).

Dans les expérimentations que nous avons menées, les phases PR ont été mises en œuvre en s'inspirant directement de l'approche d'Instruction par les Pairs (IP) introduite dans [Mazur, 1997]. Les études expérimentales présentées dans [Crouch and Mazur, 2001] montrent que l'IP est une approche d'évaluation formative tirant parti des interactions sociales ayant un impact positif sur l'engagement des étudiants et les résultats d'apprentissage.

Nos expérimentations de phase PR ont été orchestrées avec la plate-forme web elaastic [4] (anciennement Tsaap-Notes) proposant un protocole d'IP étendu [Silvestre et al., 2015]. La plate-forme elaastic permet à l'enseignant de poser des questions aux étudiants et de gérer un processus en 3 étapes :

— Pour chaque question, chaque étudiant, à l'aide d'un dispositif connecté (tablette, smartphone ou ordinateur portable), fournit une réponse (choix d'un item, argumentation au format texte libre) au système. L'enseignant dispose d'une interface l'informant en temps réel du nombre de réponses collectées.
— Sur la base de cette information, l'enseignant peut démarrer la deuxième étape du processus : le système demande à chaque apprenant d'étudier et d'évaluer jusqu'à cinq contributions apportées par les autres étudiants. Durant cette étape, les étudiants peuvent éventuellement modifier leur réponse initiale et la soumettre de nouveau à la plate-forme.
— L'enseignant peut décider de la publication du résultat en fonction du nombre d'étudiants ayant participé à la deuxième étape. Après publication, l'enseignant et les étudiants ont accès à la liste de toutes les contributions ordonnées de la mieux notée à la moins bien notée. L'enseignant et les étudiants ont alors l'opportunité d'échanger sur les résultats observés.

Nous avons utilisé elaastic pour demander aux étudiants de défendre leurs solutions. Ils devaient fournir des portions de code et expliquer leurs choix d'implantation. Ainsi les étudiants partagent donc leurs codes mais aussi leurs motivations argumentées par écrit pour choisir une solution plutôt qu'une autre et disposent de la possibilité d'améliorer leurs solutions à la lumière des confrontations de points de vue.

En terme de mise en œuvre, contrairement aux phases TDD qui demandent un taux d'encadrement raisonnable pour pouvoir assister les étudiants confrontés à des difficultés (typiquement, dans nos expérimentations, un enseignant pour 20 étudiants), les phases de PR sont légères en terme de logistique et peuvent être menées avec des groupes larges : nos expérimentations nous ont amené à effectuer des PR avec une cinquantaine d'étudiants en simultané.

4. `https://elaastic.irit.fr/elaastic-questions`

Alternativement à l'utilisation d'elaastic, d'autres modalités de PR sans supervision de l'enseignant peuvent être pratiquées. Par exemple, les étudiants peuvent être organisés en petits groupes dans lesquels ils sont invités à se présenter entre eux leur solution et à discuter des différences aperçues. Une autre possibilité peut émerger du fait que les étudiants effectuent leur développement au sein d'un dépôt git individuel ; le mécanisme de *pull request* peut être exploité pour mener une activité de revue de code par les pairs.

La structuration de scénarios pédagogiques en une succession de cycles formés d'une phase TDD suivie d'une phase PR fait naître de nombreuses questions :
— Comment avoir une idée de la progression des étudiants ?
— Comment identifier des points de blocage communs à plusieurs étudiants ?
— Une phase PR n'a de sens que si les étudiants ont achevé la phase TDD associée et ont donc des solutions à comparer. A quel moment peut-on déclencher une phase de PR ?
— L'objectif des phases de PR est notamment de consolider les connaissances acquises lors des phases de TDD. Peut-on évaluer l'efficacité d'une telle phase avant l'évaluation sommative ?

L'étude de ces questions nous a mené à développer la plate-forme Git4School que nous présentons dans la section suivante.

3 Git4School : un tableau de bord pour le suivi des étudiants et l'orchestration de scénarios pédagogiques

Une hypothèse initiale de travail indique que chaque étudiant se voit attribuer un dépôt *git* dans lequel il réalise un *commit* à chaque fin d'exercice. L'obtention d'un tel dépôt est facilitée par l'initiative *Github Classroom* [5] qui automatise la création des dépôts *git* avec des droits d'accès paramétrables tout en y incluant un code de démarrage.

La plate-forme Git4School [6] est une application web disponible à toute personne disposant d'un compte Github. Son code source [7] est publié sous licence Apache 2.0. Elle exploite l'API REST de Github [8] pour extraire des informations sur les *commits* effectués par les étudiants au sein de leurs dépôts. Git4School est un tableau de bord, à destination des enseignants, permettant de visualiser la progression des étudiants et facilitant l'orchestration des différentes phases du protocole défini dans la section précédente.

Les deux seules hypothèses de travail conditionnant l'utilisation de Git4School sont d'une part, l'utilisation de Github par les étudiants et, d'autre part, le respect de la bonne pratique consistant à effectuer un *commit* lors de chaque achèvement majeur dans le travail tel que la terminaison d'un exercice. Git4School est donc très faiblement invasif d'un point de vue méthodologique et aussi en

5. `https://classroom.github.com/`
6. `https://git4school.firebaseapp.com/`
7. `https://github.com/git4school/git4school-visu`
8. `https://developer.github.com/v3/`

terme d'outillage : la plate-forme peut être utilisée quelque soit l'environnement de travail et le langage de programmation.

L'enseignant, une fois authentifié sur Git4School, fournit un fichier au format JSON contenant les informations suivantes :

— La liste des dépôts Github des étudiants ;
— Une liste d'identifiants pour les exercices qui sont ensuite recherchés dans les messages de *commits* des étudiants. La présence d'un identifiant est interprétée comme la terminaison de cet exercice par un étudiant lors d'une phase TDD ;
— La date et la durée des séances encadrées ;
— Pour chaque exercice, la date à laquelle a lieu la phase PR le concernant ainsi que la date de la publication de sa correction. Dans la suite, on parlera de jalon pour évoquer ces événements.

Ensuite, l'enseignant a à sa disposition trois graphiques :

— **Vue d'ensemble** (figure 2) : permet de visualiser pour chaque étudiant, en ordonnée, l'ensemble de ses *commits* au cours du temps, en abscisse. Sont repérés aussi les périodes correspondant à des séances encadrées, à la réalisation de phase de PR et à la publication d'une correction. Les *commits* sont modélisés par des formes de couleur différente selon s'il sont effectués avant ou après un jalon (phase PR ou publication de la correction).

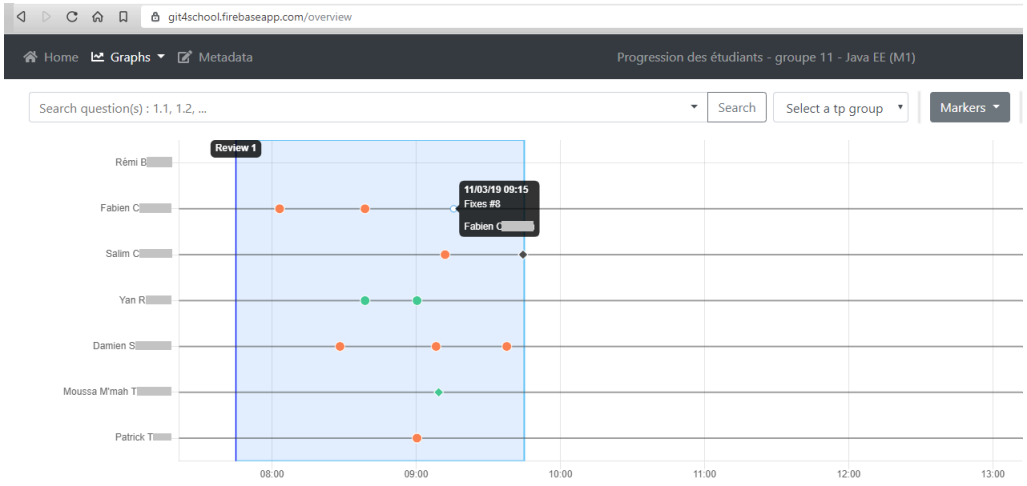

Figure 2. Visualisation de la progression de l'ensemble dans Git4School

— **Visualisation des *commits* typés** (figure 3) : permet de visualiser sous la forme d'un diagramme en bâton, pour chaque étudiant, le nombre de ses *commits* par rapport aux jalons (commits avant PR, avant publication de la correction, etc) ainsi que l'identifiant du dernier exercice traité.

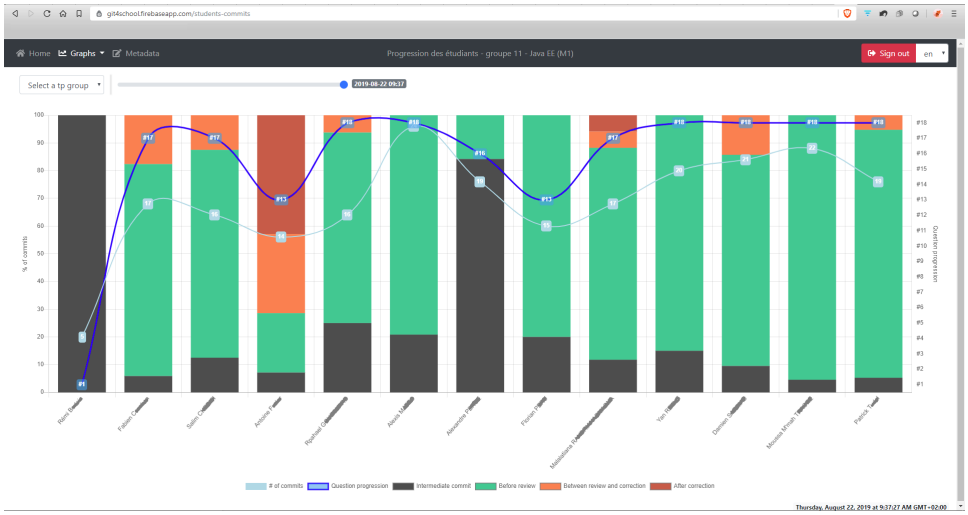

Figure 3. Visualisation du type des *commits* de chaque étudiant dans Git4School

— **Visualisation de l'accomplissement par exercice** (figure 4) : permet de visualiser sous la forme d'un diagramme en bâton, pour chaque exercice, le nombre d'étudiant l'ayant terminé et dans quelle condition (avant ou après la phase de PR, après la publication de la correction).

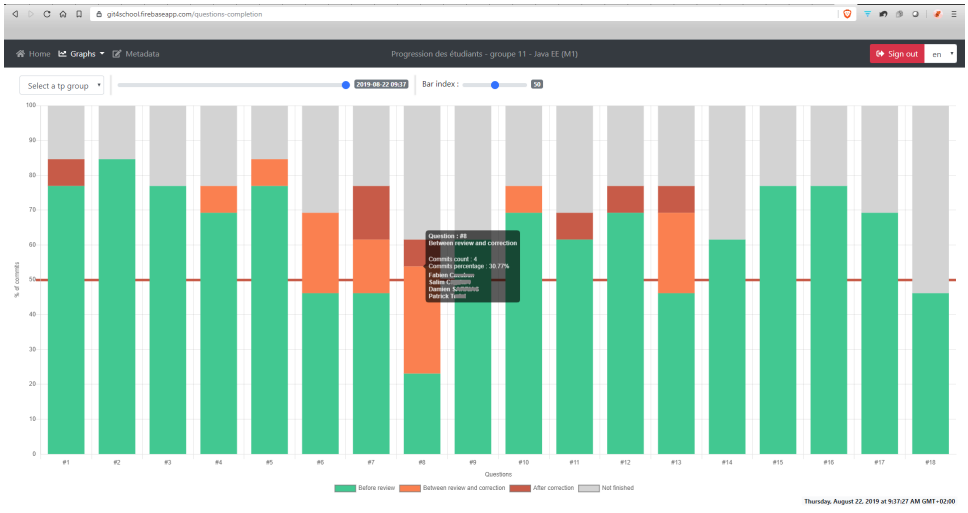

Figure 4. Visualisation de l'accomplissement par exercice dans Git4School

Les deux premiers graphiques permettent d'obtenir des informations sur la façon dont chaque étudiant travaille : comment se positionne t-il par rapport au reste de ses camarades ? s'il est en retard, essaie t-il de compenser en traitant des exercices en dehors des séances encadrées ? Tire-t-il profit des phases de PR ou de la publication de la correction pour achever les exercices sur lesquels il a des difficultés ?

Le troisième graphe est utile pour l'orchestration des phases du protocole : si suffisamment d'étudiants ont achevé l'exercice, il peut être pertinent de déclencher la phase de PR. Une autre utilisation possible de ce graphe concerne les exercices identifiés comme bloquants car terminés par peu ou pas d'étudiants ; une remédiation peut être effectuée au cours d'une phase de PR dont l'impact peut être visualisé par la couleur orange dans le diagramme en bâton qui indique le pourcentage d'étudiants ayant achevé un exercice après la phase de PR. L'enseignant dispose alors d'un retour sur sa tentative pédagogique de déblocage.

4 Discussions et travaux connexes

Alignements avec le SWEBOK et le curriculum de l'ACM/IEEE. Le protocole d'apprentissage introduit dans cet article permet d'initier précocement les étudiants à plusieurs pratiques professionnelles référencées dans le SWEBOK. Les phases TDD sont très formatrices pour le domaine de connaissance *« software construction tools »* ([Bourque and Fairley, 2014], chapitre 3) par l'utilisation intensive d'un environnement de développement, d'un *framework* de test, d'un gestionnaire de version *git* et d'un *debugger*.

Les aspects méthodologiques agiles ([Bourque and Fairley, 2014], chapitre 9) sont centraux dans le protocole avec l'idée de procéder par cycles courts de développement dans une démarche itérative centrée sur les tests ; de plus, les activités sociales des phases PR forment aux processus collaboratifs de développement ([Bourque and Fairley, 2014], chapitre 8 et 10).

Concernant le curriculum de l'ACM/IEEE, comme déjà mentionné lors de l'étude du tableau 2, le protocole grâce à l'ajout des phases PR offre une meilleure couverture des objectifs d'apprentissage du curriculum.

Engagement des étudiants. L'engagement des étudiants est un levier essentiel pour améliorer le processus d'apprentissage. Le cadre ICAP [Chi and Wylie, 2014] décrit quatre modes d'engagement des étudiants : passif, actif, constructif et interactif. En mode passif, les étudiants reçoivent directement les instructions de l'enseignant pendant les cours ou regardent une vidéo sans autre activité que l'attention à l'écran. Lorsque l'attention donnée est soutenue par une action motrice ou une manipulation physique, on dit que les étudiants sont en mode actif. Lorsque les élèves créent ou produisent du matériel supplémentaire, ils s'engagent de manière constructive. Enfin, lorsque les élèves interagissent avec d'autres (pairs, tuteurs, enseignants) pour construire de nouveaux supports, ils

sont ensuite engagés dans le mode interactif. D'après des mesures d'apprentissage, le mode interactif subsume le mode constructif qui subsume le mode actif subsumant le mode passif.

Une caractéristique majeure du protocole est qu'il propose un environnement dans lequel les étudiants basculent continuellement entre un mode constructif et un mode interactif tout au long des cycles répétés des phases TDD et PR. Ils sont donc toujours placés aux deux niveaux d'engagement les plus élevés, maximisant ainsi leurs chances d'améliorer leurs compétences en programmation.

Travaux connexes. L'apprentissage dirigé par les tests (TDL pour *Test Driven Learning*) a été introduit dans [Janzen and Saiedian, 2006] pour relever le défi d'apprendre à programmer en utilisant les tests afin d'expliciter l'usage et le comportement de portions de codes. Les résultats des expérimentations montrent que les étudiants gagnent en compréhension lorsqu'ils sont en situation d'apprentissage dirigée par les tests [Janzen and Saiedian, 2008] et qu'ils améliorent leurs performances aux évaluations [Hilton and Janzen, 2012].

Plusieurs travaux d'extraction de métriques à partir des logs d'un gestionnaire de version existent dans la littérature (voir [Mittal and Sureka, 2014] pour une vue d'ensemble) mais contrairement à notre approche qui analyse des messages de *commits* qui portent des informations sémantiques (la terminaison d'un exercice bien identifié), les travaux s'intéressent plutôt à des informations sur les *commits* de type volumétrie, fréquence, temporalité relative à une date limite ou analysent la collaboration entre membres d'un même dépôt.

Concernant Git4School, de nombreux tableaux de bord de suivi des apprentissages existent [Charleer et al., 2014,Putra et al., 2018]. Notre approche est originale par son aspect non-invasif et le fait qu'elle soit dédiée à la plate-forme Github.

5 Conclusion : pistes pour des travaux futurs

Le protocole rapporté ici a été raffiné au cours de deux années d'expérimentations. Arrivé maintenant à un état stable, les données d'apprentissage actuellement collectées sont fidèles à l'esprit du protocole. Le premier travail que nous mènerons à court terme va consister à analyser statistiquement ces données pour vérifier si les résultats aux examens se sont améliorés depuis l'établissement du protocole.

De même, peut-on corréler le comportement de l'étudiant au sein de son dépôt *git* et ses résultats lors des examens ? Le cas échéant, la mise en place d'un système d'alerte permettant d'anticiper un potentiel échec de l'étudiant et d'adapter en conséquence le contenu d'une phase PR constituerait une piste future de travail.

Références

[Beck, 2003] Beck, K. (2003). *Test-driven development : by example.* Addison-Wesley Professional.

[Bergeat, 2017] Bergeat, M. (2017). Les tensions sur le marché du travail en 2017. https://dares.travail-emploi.gouv.fr/IMG/pdf/2017-056.pdf. Rapport DARES Indicateurs, Ministère du travail.

[Biggs, 1996] Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher education*, 32(3) :347–364.

[Bourque and Fairley, 2014] Bourque, P. and Fairley, R. E., editors (2014). *SWEBOK : Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society.

[Charleer et al., 2014] Charleer, S., Santos, J., Klerkx, J., and Duval, E. (2014). Improving teacher awareness through activity, badge and content visualizations. In *New Horizons in Web Based Learning*, volume 8699, pages 143–152.

[Chi and Wylie, 2014] Chi, M. T. and Wylie, R. (2014). The icap framework : Linking cognitive engagement to active learning outcomes. *Educational Psychologist*, 49(4) :219–243.

[Colin et al., 2015] Colin, J.-F., Aboudara, S., Jolly, C., Lainé, F., Argouarc'h, J., and Bessière, S. (2015). Les métiers en 2022. https://www.strategie.gouv.fr/sites/strategie.gouv.fr/files/atoms/files/fs_rapport_metiers_en_2022_27042015_final.pdf. Rapport de la DARES, Ministère du travail.

[Crouch and Mazur, 2001] Crouch, C. H. and Mazur, E. (2001). Peer instruction : Ten years of experience and results. *American journal of physics*, 69(9) :970–977.

[Hilton and Janzen, 2012] Hilton, M. and Janzen, D. S. (2012). On teaching arrays with test-driven learning in webide. In *Conference on Innovation and technology in computer science education*, pages 93–98. ACM.

[Janzen and Saiedian, 2008] Janzen, D. and Saiedian, H. (2008). Test-driven learning in early programming courses. In *ACM SIGCSE Bulletin*, volume 40, pages 532–536. ACM.

[Janzen and Saiedian, 2006] Janzen, D. S. and Saiedian, H. (2006). Test-driven learning : intrinsic integration of testing into the cs/se curriculum. In *ACM SIGCSE Bulletin*, volume 38, pages 254–258. ACM.

[Mazur, 1997] Mazur, E. (1997). Peer instruction : getting students to think in class. In *AIP Conference Proceedings*, pages 981–988.

[Mittal and Sureka, 2014] Mittal, M. and Sureka, A. (2014). Process mining software repositories from student projects in an undergraduate software engineering course. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 344–353. ACM.

[Putra et al., 2018] Putra, F., Santoso, H., and Aji, R. (2018). Evaluation of learning analytics metrics and dashboard in a software engineering project course. *Global Journal of Engineering Education*, 20(3) :171–180.

[Sahami et al., 2013] Sahami, M., Roach, S., and ACM/IEEE-CS Joint Task Force on Computing Curricula (2013). Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press.

[Silvestre et al., 2015] Silvestre, F., Vidal, P., and Broisin, J. (2015). Reflexive learning, socio-cognitive conflict and peer-assessment to improve the quality of feedbacks in online tests. In *Design for Teaching and Learning in a Networked World*, pages 339–351. Springer.