Étude exploratoire de la pertinence des traces pour l'évaluation d'un projet de programmation

Christopher Marshall-Breton[1 [0009−0007−2204−9613]], Maude Pupin[1 [0000−0003−3197−0715]], Yvan Peter[1 [0000−0002−1145−357X]] et Mirabelle Nebut[1]

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
christopher.marshallbreton.etu@univ-lille.fr

Résumé. L'évaluation des acquis de programmation peut, en partie, être automatisée. De nombreux outils existent mais présentent cependant deux limites majeures : leur transférabilité à d'autres contextes pédagogiques, et la pertinence pour l'évaluation d'un projet. Nous avons donc un ensemble de traces remontées par ces différents outils et une difficulté à les appliquer à l'évaluation des projets. L'objet de cet article est de proposer une étude exploratoire pour étudier la faisabilité d'une évaluation entièrement automatique pour un projet de programmation. Nous nous concentrerons sur les débutants en programmation.

Mots-clé: Évaluation · Outils d'évaluation automatique · Projet de programmation · Traces d'apprentissage · Enseignement de la programmation.

Abstract. The evaluation of programming knowledge can be partly automated. Many tools exist but have two major limitations: Their transferability to other educational contexts, and the relevance for the evaluation of a project. We therefore have a set of traces returned by these different tools and a difficulty in applying them to the evaluation of projects. The purpose of this article is to propose an exploratory study to study the feasibility of a fully automatic evaluation for a programming project. We will focus on beginners.

Keywords: Evaluation · Automated Assessment Tools · Developpment Project · Learning Analytics · Programming Teaching.

1 Introduction

D'après FranceTravail [4], les besoins dans le numérique sont toujours en augmentation. Le métier de développeur, en 2025, reste le plus en tension. Dans les formations post-bac, l'informatique attire, avec 10.5% des inscriptions en cycle ingénieur dans le domaine informatique [9], et les promotions ont des effectifs conséquents. Ce nombre d'étudiants entraîne deux conséquences : D'une part,

l'évaluation devient une activité chronophage, d'autre part, de multiples parcours se développent pour les mêmes objectifs de formation. Une des réponses au premier point repose sur des outils automatiques de correction. Plusieurs outils et études se sont intéressés à ces pratiques dans un cadre d'évaluation [1,10,8]. Cependant, ces outils présentent deux limitations majeures :

Ils sont d'abord liés au processus d'apprentissage (langage, concepts évalués...). La transférabilité à un nouveau contexte pédagogique est donc difficile, et pousse souvent les équipes à préférer développer un nouvel outil propre à leurs contextes, entraînant une multiplication du nombre d'outils, avec une portée limitée [11]. Ce phénomène est exacerbé par le nombre de formations différentes engendrées par le nombre important d'étudiants comme mentionné ci-dessus.

La seconde limitation identifiée est l'inadéquation apparente de ces outils pour l'évaluation des projets. En effet, les contraintes liées à l'automatisation de l'évaluation s'opposent à la créativité recherchée dans ces projets.

L'objet de cet article est de proposer une étude exploratoire pour examiner dans quelle mesure les analyses de traces utilisées par ces outils restent pertinentes sur des projets. On étudiera également la transférabilité de ces traces à un cadre de formation différent. Nous utiliserons pour 'traces' la définition de Learning Analytics établie pour la conférence LAK 2011 et adoptée par la Society for Learning Analytics Research (SoLAR) [3], à savoir 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 déroule.

2 Outils existants et traces

L'évaluation des étapes d'apprentissage de la programmation est complexe. Les notions à enseigner sont difficiles à identifier ou restent discutées quand elles le sont [7,13]. Aux premiers pas de l'apprentissage, on se concentre sur des exercices simples, isolés, avec un focus sur le résultat. Plus on progresse, plus les critères de validation se multiplient. Généralement, le processus d'apprentissage présente les différents concepts de la programmation par des exercices ciblés, puis permettent aux étudiants de les mettre en pratique par des projets plus larges.

De nombreux outils d'évaluation automatique existent, se concentrant majoritairement sur une succession d'exercices évalués par tests unitaires. Ces exercices sont propres à une progression pédagogique et sont généralement liés au cadre d'enseignement (utilisation d'un langage donné, notions absentes de certains exercices car pas encore abordées, *etc.*) [1,10,8]. Ce cadre restreint permet une évaluation automatique plus simple, car ces exercices sont justement très cadrés.

De ces exercices, on peut tirer un ensemble de traces de natures très différentes. L'analyse statique, sans exécution de code, permet par exemple d'avoir des informations sur le respect des conventions de nommage, sur la complexité cyclométrique, le nombre de commentaires...). L'analyse dynamique permet quant à elle d'avoir de informations sur l'exécution du code (validation de tests uni-

taires, complexité temporelle ou spatiale, erreurs de compilation/exécution...). Les traces que l'on peut remonter dépendent évidement de l'outil utilisé. C. Douce *et al.* [2] classent les outils d'évaluation automatique selon trois générations : les systèmes d'évaluation précoces (dès 1960) donnant des informations rudimentaires sur l'exécution du programme et difficiles à mettre en place, les systèmes axés sur des outils qui élargissent les possibilités de tests automatiques, augmentent le nombre de métriques mesurées et facilite le déploiement, puis les systèmes orientés web. Ces derniers, en plus de faciliter la soumission des travaux et de permettre un suivi plus personnalisé, offrent un contrôle plus précis sur le contexte d'évaluation, ce qui permet d'isoler des traces plus variées (temps passé sur un code, touches pressées...).

3 Projets de programmation

Dans une séquence pédagogique classique, les exercices sont suivis par un ou plusieurs projets, qui permettent une mise en pratique plus libre des différentes notions apprises. Les pédagogies par projet sont aujourd'hui très largement étudiées et appliquées [5]. Dans l'apprentissage de la programmation, ces projets permettent aux étudiants de mobiliser leurs acquis avec une plus grande autonomie que celle limitée par le cadre restreint des exercices isolés.

Pourtant, la liberté offerte par ces projets rend difficile leur évaluation automatique. Celle-ci se fait donc généralement manuellement par l'enseignant, par revue de code, soutenance, ou rapport.

Les projets sont également essentiels par la mise en situation et l'autonomie nécessaire à leur réalisation. Plus que la simple restitution des éléments appris, ils permettent souvent de confronter les étudiants à une situation nouvelle, et d'évaluer comment ils y font face (par exemple : utilisation d'une bibliothèque graphique sans explication préalable).

Ces projets permettent enfin de se rapprocher du contexte professionnel. Ils permettent notamment de présenter des concepts plus larges, qui n'ont que peu de sens pour des exercices isolés (gestionnaire de version, batterie de tests, intégration...)

Nous proposons donc la question de savoir si un ensemble de traces remontées dans un cadre restreint, reste pertinent sur un cadre plus large (projet) et si l'on peut étendre ce projet à une formation différente.

4 Transposition des traces à un projet : cadre d'étude

L'objectif premier de cette étude est d'identifier les traces les plus pertinentes. Pour cela, il est nécessaire de produire des traces de natures différentes.

De nombreux outils d'évaluation automatique existent, les plus connus étant Online Judge [12], ou CodingGame pour le secteur privé. Comme précisé plus haut, le nombre d'outils existant rend difficile le choix d'un en particulier, chacun ayant ses spécificités [14].

En tant que première expérience, nous travaillerons avec l'outil que notre public utilise actuellement. Il s'agit d'un plugin de l'éditeur Thonny, développé par l'Université de Lille.[6] Nous utiliserons également le langage pratiqué par les étudiants pendant ce semestre : Python.

Les traces collectées seront celles permises par l'outil, à savoir en majorité des tests fonctionnels, des informations sur la temporalité de la création des fonctions, ou de l'analyse statique sur le code rendu.

La réflexion se fera surtout sur l'analyse de ces traces. Nous voulons donc d'une part savoir quelles traces peuvent être facilement remontées, d'autre part nous intéresser à l'analyse et à la pertinence de celles-ci dans un contexte plus large.

Prenons un exemple plus concret : Une génération de docstring. Pour un exercice isolé simple, nous pouvons extraire la docstring d'une fonction unique à signature imposée demandée par l'énoncé, et vérifier sa pertinence. Dans un projet plus large et moins directif, il devient compliqué de remonter et analyser les docstrings de toutes les fonctions. La question devient donc : comment peut-on adapter le projet pour conserver la pertinence de cette trace, ou est-elle inutile car trop générale et donc difficilement exploitable ? Cette situation s'applique à de nombreuses autres traces, ayant chacune des problématiques, d'où l'objectif de cette étude. Nous avons déjà identifié plusieurs difficultés, l'étude vise à les contourner.

Notons également que nous jugeons la "pertinence" sur deux axes : La trace est-elle exploitable dans un contexte projet ? La trace permet-elle de juger d'une compétence ?

5 Construction du projet

Pour des raisons logistiques et de cohérence avec le programme de l'année, l'évaluation durera 45 min. Nous aurons 3 exercices. Un exercice de compréhension de code, nécessitant une amélioration, un exercice de recherche d'erreur nécessitant une correction, et un exercice d'écriture de code nécessitant de produire le corps d'une fonction.

Le but de cette évaluation est de se séparer d'une organisation classique d'exercices indépendants. Pour se rapprocher d'une situation professionnelle, un contexte a été imaginé : l'étudiant est en stage dans une entreprise de développement et reçoit des mails de son manager, lui demandant de réaliser certaines tâches.

Chaque exercice est présenté comme une de ces tâches. Il contient en en-tête le mail commenté, puis un morceau de code. Contrairement aux exercices de l'année, le code fourni contient volontairement des oublis de bonnes pratiques. Il contient par exemple un oubli dans les annotations de types, ou une absence de cohérence dans le nom des fonctions (franglais).

Nous ne détaillerons pas ici les différents exercices, mais nous pouvons lister quelques traces que nous pouvons en extraire (Tableau 1).

Tableau 1. Exemples de traces remontées par les 3 exercices.

N°	Descriptif de l'exercice	Objectif général	Traces
1	Ajout d'option à un code préexistant	Mise à niveau de code, Compréhension de code	Fonctionnel (Test unitaire), Mise à jour documentation (Analyse statique), Homogénéisation du code (Analyse statique)
2	Correction d'erreur dans un code préexistant	Correction de code, Compréhension de code	Fonctionnel sur borne de boucle (Test unitaire), Fonctionnel sur ré initialisation de compteur (Test unitaire), Optimisation de code (Analyse statique)
3	Écriture d'une fonction dans un environnement préexistant	Écriture de code	Navigation dans les fichiers (Logs), Temps de résolution (Logs), Utilisation des données test (Analyse statique), Fonctionnel (Test unitaire)

Cette base nous permettra dans un second temps de mettre en place une épreuve projet beaucoup plus large, afin d'étudier les traces isolées dans les exercices précédents dans un contexte de projet.

6 Conclusion

Les outils d'évaluation automatique sont nombreux et sont souvent limités à un contexte d'enseignement très particulier. C'est pourtant ce cadre restreint qui permet de tirer relativement simplement diverses traces d'apprentissage à partir des exercices d'étudiants.

Nous allons dans un premier temps récupérer un ensemble varié de traces à partir d'exercices simulant une situation professionnelle. Ces traces seront ensuite analysées pour évaluer leur pertinence dans un contexte plus large, à savoir un projet de programmation.

Remerciements. This work is funded by the Interreg France-Wallonie-Vlaanderen *Open Badges for IT* project.

References

1. Combéfis, S.: Automated Code Assessment for Education: Review, Classification and Perspectives on Techniques and Tools. Software **1**(1), 3–30 (Feb 2022). https://doi.org/10.3390/software1010002, https://www.mdpi.com/2674-113X/1/1/2
2. Douce, C., Livingstone, D., Orwell, J.: Automatic test-based assessment of programming: A review. J Educ Resour Comput **5** (Sep 2005). https://doi.org/10.1145/1163405.1163409

3. Ferguson, R.: Learning analytics: drivers, developments and challenges. International Journal of Technology Enhanced Learning **4**(5-6), 304–317 (2012)
4. France-Travail: Les métiers du numérique : Tour d'horizon des enjeux et des opportunités, https://www.francetravail.org/accueil/actualites/infographies/2025/les-metiers-du-numerique-tour-d'horizon-des-enjeux-et-des-opportunites.html?type=article
5. Kokotsaki, D., Menzies, V., Wiggins, A.: Project-based learning: A review of the literature. Improving schools **19**(3), 267–277 (2016)
6. Marvie-Nebut, M., Peter, Y.: Apprentissage de la programmation python-une première analyse exploratoire de l'usage des tests. In: Atelier «Apprendre la Pensée Informatique de la Maternelle à l'Université», dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). pp. 1–8 (2023)
7. Medeiros, R.P., Ramalho, G.L., Falcão, T.P.: A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. IEEE Transactions on Education **62**(2), 77–90 (May 2019). https://doi.org/10.1109/TE.2018.2864133, https://ieeexplore.ieee.org/document/8447543/?arnumber=8447543, conference Name: IEEE Transactions on Education
8. Messer, M., Brown, N.C.C., Kölling, M., Shi, M.: Automated Grading and Feedback Tools for Programming Education: A Systematic Review. ACM Transactions on Computing Education **24**(1), 1–43 (Mar 2024). https://doi.org/10.1145/3636515, https://dl.acm.org/doi/10.1145/3636515, publisher: Association for Computing Machinery (ACM)
9. Ministere-ESR: Les effectifs inscrits en cycle ingénieur en 2023-2024 (Jun 2024), https://www.enseignementsup-recherche.gouv.fr/fr/les-effectifs-inscrits-en-cycle-ingenieur-en-2023-2024-96520
10. Paiva, J.C., Leal, J.P., Figueira, A.: Automated Assessment in Computer Science Education: A State-of-the-Art Review. ACM Transactions on Computing Education **22**(3), 1–40 (Sep 2022). https://doi.org/10.1145/3513140, https://dl.acm.org/doi/10.1145/3513140
11. Pettit, R., Prather, J.: Automated assessment tools: too many cooks, not enough collaboration. J. Comput. Sci. Coll. **32**(4), 113–121 (Apr 2017)
12. Revilla, M.A., Manzoor, S., Liu, R.: Competitive learning in informatics: The uva online judge experience. Olympiads in Informatics **2**(10), 131–148 (2008)
13. Robins, A., Rountree, J., Rountree, N.: Learning and Teaching Programming: A Review and Discussion. Computer Science Education **13**(2), 137–172 (Jun 2003). https://doi.org/10.1076/csed.13.2.137.14200, https://www.tandfonline.com/doi/full/10.1076/csed.13.2.137.14200
14. Wasik, S., Antczak, M., Badura, J., Laskowski, A., Sternal, T.: A survey on online judge systems and their applications. ACM Computing Surveys (CSUR) **51**(1), 1–34 (2018)