Identification des lacunes dans le savoir conceptuel et stratégique dans l'apprentissage de la programmation — *Extended abstract*

Jean-Philippe Pellet[0000−0001−7559−397X] et Patrick Wang[0000−0003−3117−8189]

Haute école pédagogique du canton du canton de Vaud, Lausanne, Suisse
{jean-philippe.pellet, patrick.wang}@hepl.ch

L'apprentissage de la programmation est un sujet largement étudié dans la littérature et de nombreux articles (e.g., [1,2]) cherchent à mettre en avant les principales difficultés que les élèves peuvent rencontrer au cours de cet apprentissage. Ces difficultés se manifestent souvent lorsqu'il s'agit de concevoir une solution algorithmique à un problème donné et/ou lorsqu'il s'agit de traduire cette solution algorithmique en un programme. Dans [2], les auteurs expliquent que ces erreurs peuvent provenir de lacunes dans le savoir *syntaxique*, *conceptuel*, ou *stratégique*. Pour un enseignant, il est *a priori* facile de repérer et d'évaluer les lacunes dans le savoir syntaxique, les messages d'erreurs en console offrant une aide précieuse ici. Mais l'identification précise des points de blocage dans le savoir conceptuel et stratégique est plus complexe.

Dans cette proposition de poster, nous nous intéressons à cette question de l'évaluation du savoir conceptuel et stratégique. La question de recherche qui motive ce travail est la suivante : « Comment identifier et évaluer, dans les productions des élèves, des lacunes dans le savoir conceptuel et stratégique ? ».

Nous avons à cet effet construit un dispositif d'évaluation notamment basé sur des « études de cas » ayant pour objectif de mettre en évidence le savoir conceptuel et stratégique mobilisé durant ces activités. L'idée de base est l'explicitation de l'identification et de la juxtaposition des éléments clés du langage nécessaires pour résoudre une tâche donnée, avant de passer à l'écriture de code.

La question de la stratégie de conception d'un algorithme ou de construction d'un programme n'est pas nouvelle. En effet, Wirth [4] proposait déjà en 1971 la méthode du *stepwise refinement*, qui consiste à formuler une solution dont les instructions sont tout d'abord assez larges, puis à les raffiner jusqu'à obtenir des instructions dont le niveau de précision équivaudrait à celui pouvant être obtenu par un langage de programmation. Cependant, Rist [3] explique qu'un expert, contrairement au novice, sera capable d'alterner (ponctuellement et lorsque nécessaire) entre des approches *top-down* (similaires au *stepwise refinement*) et *bottom-up* lors de la construction d'un programme.

Notre dispositif cherche donc à rendre compte de la capacité des élèves à concevoir une solution algorithmique tout en permettant la mise en évidence des différences de niveau de maîtrise dans les savoirs conceptuels et stratégiques. Pour cela, nous avons conçu une activité en deux temps autour d'études de cas. Tout d'abord il est demandé aux élèves de formuler en langage naturel un algorithme de résolution possible. La présence de constructions algorithmiques et

le niveau de précision des instructions sont des indicateurs que nous souhaitons trouver dans les réponses des élèves pour répondre à notre question de recherche. Durant le second temps, les élèves implémentent leur algorithme. Ce travail vise à expliciter un éventuel écart pouvant exister entre une solution algorithmique formulée en langage naturel et son implémentation directe, écart pouvant suggérer des difficultés à aller « jusqu'au bout » du *stepwise refinement*. En guise d'illustration, voici ci-après un exemple d'étude de cas.

Exemple. L'élève, après 4 semaines d'apprentissage de la programmation avec Python, reçoit la consigne : « Pour la tâche suivante, indiquer les éléments du langage ou les structures dont nous aurons besoin. Pour chaque élément, indiquer (a) combien de fois on y fera référence dans le programme et pourquoi, et (b) combien de fois il sera "utilisé" pendant l'exécution du programme. *Tâche :* proposer à l'utilisateur d'entrer une série de notes entre 1 et 6, puis à la fin, afficher la moyenne. » La réponse attendue, dont la forme a été travaillée en classe, ressemble à ceci (étant entendu que les listes Python sont encore inconnues à ce stade). *Il nous faut :*

— *Une* ***boucle****, car il s'agit d'introduire des nombres de manière répétée. Il y aura une seule boucle, dont le corps pourra être exécuté plusieurs fois ;*

— *Un* ***input() avec conversion en int*** *pour entrer les nombres, qui sera visible une fois dans le programme mais qui sera situé dans la boucle et donc exécuté potentiellement plusieurs fois aussi ;*

— *Une* ***condition*** *pour vérifier que soit entre 1 et 6, après chaque input() – aussi une fois dans le programme, mais exécuté dans la boucle ;*

— *Des* ***variables auxiliaires****, une pour faire la somme des valeurs entrées, l'autre pour compter le nombre de valeurs entrées. Elles sont initialisées avant la boucle, mises à jour dans la boucle, et utilisées après la boucle pour calculer la moyenne.*

L'analyse des données n'est actuellement pas terminée mais serait, en cas d'acceptation, complétée pour le poster selon les pistes principales suivantes : (a) capacité à identifier les structures algorithmiques nécessaires et (b) écart entre formulation textuelle et écriture du programme.

Références

1. Du Boulay, B. : Some Difficulties of Learning to Program. Journal of Educational Computing Research **2**(1), 57–73 (Feb 1986). https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9
2. Qian, Y., Lehman, J. : Students' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review. ACM Transactions on Computing Education **18**(1), 1 :1–1 :24 (Oct 2017). https://doi.org/10.1145/3077618
3. Rist, R.S. : Schema creation in programming. Cognitive Science **13**(3), 389–414 (1989)
4. Wirth, N. : Program development by stepwise refinement. Communications of the ACM **14**(4), 221–227 (1971)