Actes de l'atelier "Apprendre la Pensée Informatique de la Maternelle à l'Université"
Brest, France, June 13, 2023

Expérimentation du *pair-programming* en spécialité NSI en classe de première pour l'acquisition de compétences en programmation

Sophie Chane-Lune[1] Christophe Declercq[1] Sébastien Hoarau[1]
(1) Laboratoire d'informatique et de mathématiques, Université de la Réunion
Sophie.Chane-Lune@ac-reunion.fr, christophe.declercq@univ-reunion.fr,
seb.hoarau@univ-reunion.fr

RÉSUMÉ

Le *pair-programming* est une modalité d'enseignement pratique de la programmation où les deux élèves ont des rôles différents : le *pilote* programme pendant que le *co-pilote* analyse, commente, teste le travail de son binôme. L'approche par compétences permet de distinguer finement les compétences développées par les élèves et en particulier d'identifier les compétences spécifiques à chaque rôle dans une activité en *pair-programming*. En classe de première, le programme de la spécialité NSI mêle des notions d'algorithmique et de programmation avec un focus particulier sur les méthodes de programmation incluant spécification, mise au point et tests. Notre travail a consisté à expérimenter une séquence d'activités sur ces notions, pour montrer l'intérêt combiné du *pair-programming* et de l'approche par compétences.

ABSTRACT

Experimentation of pair-programming in college for the acquisition of programming skills

Pair-programming is a method of practical teaching of programming where the two students have different roles: the pilot programs while the co-pilot analyzes, comments, tests the work of his pair. The skills-based approach makes it possible to finely distinguish the skills developed by the students and in particular to identify the skills specific to each role in a pair-programming activity. In college year 12, computer science curriculum combines notions of algorithms and programming with a particular focus on programming methods including specification, development and testing. Our work consisted in experimenting with a sequence of activities on these notions, to show the combined interest of pair-programming and skills-based approach.

MOTS-CLÉS : Numérique et science informatique, compétences, collaboration, apprentissage, programmation.

KEYWORDS: Computer science education, teaching, pair-programming, skills.

1 Introduction

Au-delà du cadre de l'enseignement, le *pair-programming* est un style de programmation où deux programmeurs collaborent côte à côté sur la même machine, sur le même code, le même algorithme ou les mêmes tests (Williams & Kessler, 2003). Apparu très tôt chez les programmeurs en activité (les premiers retours datent de la fin des années 60, bien avant de s'appeler *pair-programming*), ce style de programmation fait son apparition dans le monde de l'éducation dans les années 2000 et depuis, a fait l'objet de nombreuses recherches qui montrent son intérêt, voir par exemple (Hanks et al., 2011), (Tunga & Tokel, 2018).

Néanmoins, les études concernent essentiellement le *pair-programming* dans le cadre de cours de programmation dans le supérieur et peu de choses ont été proposées pour l'apprentissage des débutants au niveau du lycée.

L'utilisation d'artefacts pour outiller le *pair-programming* a été documenté en particulier par (Schümmer & Lukosch, 2009) dans ce qu'ils nomment *distributed pair-programming*. Nous proposons, dans ce cadre, un nouvel instrument très léger permettant de mettre en œuvre à moindre coût une activité de *pair-programming* dans le cadre d'un travail pratique au lycée. Cet instrument s'apparente aussi - en version très simplifiée - aux environnements de laboratoire de travaux pratiques à distance tels que Lab4ce (Broisin et al., 2015), auquel nous empruntons l'idée de partage d'une console d'un élève avec un autre élève.

Concernant l'approche par compétences en programmation, nous nous appuyons sur les travaux fondateurs de (Wing, 2006), complétés par les précisions opérationnelles apportées par (Selby & Woollard, 2014). Nous avons aussi déjà proposé de renommer en français les compétences avec les verbes : *évaluer, anticiper, décomposer, généraliser et abstraire* (Declercq, 2022) auquel nous proposons maintenant d'ajouter la compétence *modéliser* qui nous paraît fondamentale dans le choix des structures de données étudiées en spécialité NSI.

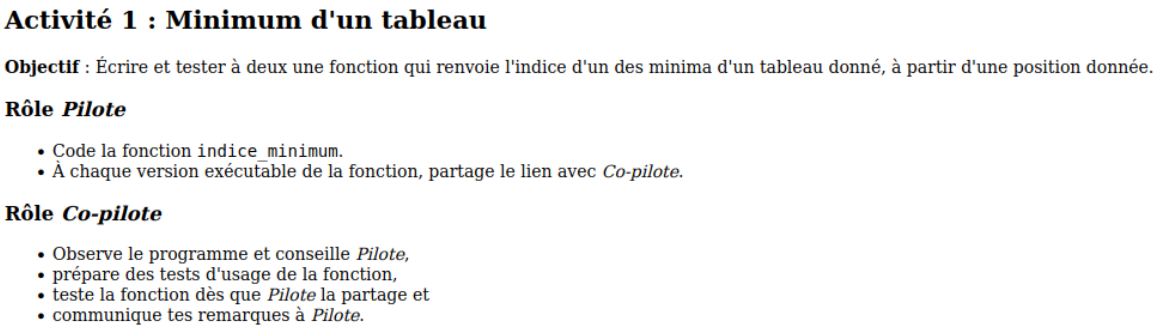
Dans cette communication, nous proposons de décrire une séquence d'activités en *pair-programming* en précisant notamment les deux rôles du binôme, ainsi que notre outil en ligne dédié à des activités de *pair-programming* ou de partage de code. Puis, nous faisons une analyse des compétences en jeu dans les activités pour chacun des rôles. Nous terminerons par une analyse de l'expérimentation réalisée en classe de première NSI et les perspectives de cette étude exploratoire.

2 Le *pair-programming*, une méthode collaborative d'apprentissage au lycée

L'idée initiale d'expérimenter cette méthode a été formulée afin de poursuivre les objectifs suivants pour les élèves :

- favoriser la réflexion en amont de l'écriture du code et de partager cette réflexion avec son binôme,
- améliorer la phase de tests des fonctions écrites,
- améliorer l'écriture de fonctions et notamment réussir à écrire des fonctions plus complexes.

Pour mettre en œuvre la méthode au lycée, l'énoncé précise aux élèves les rôles respectifs du *pilote* et du *co-pilote*. Par exemple, pour la première activité de la séance en cours d'expérimentation, l'énoncé est le suivant (voir figure 1).



Activité 1 : Minimum d'un tableau

Objectif : Écrire et tester à deux une fonction qui renvoie l'indice d'un des minima d'un tableau donné, à partir d'une position donnée.

Rôle *Pilote*

- Code la fonction `indice_minimum`.
- À chaque version exécutable de la fonction, partage le lien avec *Co-pilote*.

Rôle *Co-pilote*

- Observe le programme et conseille *Pilote*,
- prépare des tests d'usage de la fonction,
- teste la fonction dès que *Pilote* la partage et
- communique tes remarques à *Pilote*.

Figure 1: Activité 1

L'activité 2 est présentée à la section suivante (voir figure 2) dans le cadre de la présentation de notre artefact et consiste à utiliser la fonction précédente pour positionner le minimum d'un tableau en première position. La troisième activité consiste à généraliser la fonction écrite à l'activité 2 pour positionner le minimum relatif à partir d'une position donnée. La dernière activité consiste à intégrer la ou les fonctions précédentes en vue de construire le programme de tri d'un tableau par recherche du minimum.

La mise en œuvre du *pair-programming* est faite en imposant un changement de rôle à chaque activité, pour que chaque élève ait alternativement chacun des deux rôles.

3 Proposition d'un outil pour des activités en ligne de *pair-programming*

Une interface en ligne permet à l'enseignant de saisir un énoncé (en markdown), une partie de code Python fourni, puis de générer un lien pouvant être partagé avec les élèves ayant le rôle *pilote*. Les *pilotes* disposent alors, dans leur navigateur, de l'énoncé, d'un éditeur en ligne avec le code à compléter, d'une console et d'un lien permettant de partager leur travail avec leur *co-pilote* (voir figure 2).

Le *co-pilote* accède alors à l'énoncé, au code non modifiable et à une console permettant d'effectuer les tests. Le *co-pilote* ne dispose ni du bouton `Run`, ni du bouton de partage : tout ce qu'il peut faire, avec le code fourni, se déroule dans la console (voir figure 3).

Conformément aux règles du *pair-programming*, un seul élève peut modifier le code à un moment donné. Les élèves sont encouragés à échanger, le *co-pilote* ne pouvant pas modifier le code, doit l'analyser, détecter d'éventuelles erreurs par les tests et en rendre compte au *pilote*.

Les tests réalisés dans la console permettent ici de constater que le code proposé est erroné car un élément a été dupliqué et un autre perdu. La recherche, par le *co-pilote*, de l'erreur dans le code consiste à évaluer ce que le programme, fourni par le *pilote,* fait effectivement et de comparer avec ce qu'il est censé faire.

Activité 2 - Positionne le minimum dans un tableau

Cette activité est à commencer quand la fonction `indice_minimum` de l'activité 1 fonctionne (validée par *Co-pilote*). Commencer par échanger les rôles (*Pilote* ⇔ *Co-pilote*).

Objectif

Écrire et tester à deux une fonction `positionne_minimum` qui prend un tableau en paramètre et qui met en première position du tableau, la plus petite valeur de ce tableau. **Attention**, la fonction modifie bien le contenu du tableau, mais on ne doit perdre **aucune** des valeurs.

```
def positionne_minimum (tableau):
    tableau[0] = 0
```

RUN

```
Brython 3.11.2 on Netscape 5.0 (X11; Ubuntu)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Partage la console | Partage éditeur et console

Figure 2: Vue du pilote

Activité 2 - Positionne le minimum dans un tableau

Cette activité est à commencer quand la fonction `indice_minimum` de l'activité 1 fonctionne (validée par *Co-pilote*). Commencer par échanger les rôles (*Pilote* ⇔ *Co-pilote*).

Objectif

Écrire et tester à deux une fonction `positionne_minimum` qui prend un tableau en paramètre et qui met en première position du tableau, la plus petite valeur de ce tableau. **Attention**, la fonction modifie bien le contenu du tableau, mais on ne doit perdre **aucune** des valeurs.


```
def indice_minimum(tableau):
    min, i_min = tableau[0], 0
    for i in range (len(tableau)):
        if tableau[i] < min:
            min, i_min = tableau[i], i
    return i_min
def positionne_minimum (tableau):
    i = indice_minimum(tableau)
    tableau[0] = tableau[i]
```

```
Brython 3.11.2
>>> valeurs = [92, 45, 89, 34, 56, 26, 38]

>>> positionne_minimum(valeurs)
>>> valeurs
[26, 45, 89, 34, 56, 26, 38]
>>> positionne_minimum(valeurs)
>>> valeurs
[26, 45, 89, 34, 56, 26, 38]
>>> valeurs = [92, 45, 89, 34, 56, 26, 38]


>>> positionne_minimum(valeurs)
>>> valeurs
[26, 45, 89, 34, 56, 26, 38]
>>>
```

Figure 3: Vue du co-pilote

4 Approche par compétences de l'apprentissage de la programmation

Les programmes de spécialité NSI déclinent une liste de capacités attendues dans un référentiel organisé par domaines et notions. La granularité de ces capacités est assez fine ; certaines sont liées entre elles, en particulier entre les domaines programmation et algorithmique. De manière assez orthogonale, les travaux dans le cadre de *computational thinking* sur les compétences acquises à l'occasion de l'apprentissage de la programmation, identifient des compétences cognitives très générales. Nous tentons actuellement de croiser les deux approches. Le tableau suivant donne un extrait des compétences en programmation que nous avons identifiées et catégorisées. Cet extrait concerne le domaine des tableaux incluant construction, parcours et tri de ces tableaux.

Tableau 1: Compétences en programmation - extrait concernant les tableaux

	Construire un tableau et accéder à ses éléments	***Parcourir un tableau pour rechercher ou calculer des informations***	***Parcourir et modifier un tableau par des boucles imbriquées***
Évaluer	E1 : évaluer le résultat d'un programme utilisant un tableau	E2 : évaluer le résultat d'un programme itérant sur un tableau	E3 : évaluer les valeurs successives des indices avec des boucles imbriquées
Modéliser	M1 : modéliser une série d'informations par un tableau donné en extension	M2 : modéliser un traitement de tableau en choisissant d'itérer sur les indices ou les éléments	
Anticiper	A1 : anticiper les traitements à programmer pour accéder aux éléments d'un tableau par leurs indices et les modifier	A2 : anticiper un traitement nécessitant un parcours linéaire d'un tableau	A3 : anticiper un traitement de tableau nécessitant plusieurs parcours imbriqués
Décomposer			D3 : décomposer en identifiant des parcours séquentiels de tableaux
Généraliser	G1 : généraliser l'écriture d'un tableau en utilisant la compréhension		
Abstraire	X1 : abstraire les données en considérant que les éléments d'un tableau peuvent être des tableaux		X3 : abstraire un parcours séquentiel en utilisant une fonction avec un tableau en paramètre

Les compétences que nous avons retenues articulent savoirs et savoirs-faire utilisés en contexte.

4.1 Analyse a priori des compétences développées selon les rôles

Lors des activités 1, 2 et 3, on attend du *pilote* qu'il développe les compétences A1 et A2. Il est en effet en charge de l'écriture de programmes nécessitant d'itérer sur un tableau.

Le *co-pilote* lui, va apprendre à évaluer les programmes fournis, et développer ainsi les compétences E1 et E2. Il aura aussi à modéliser les données : compétence M1.

Lors de la quatrième activité (tri par recherche du minimum), le *pilote* va développer la compétence A3. Les compétences D3 et X3 ont, quant à elles, été prises en charge par l'enseignant qui a proposé un énoncé guidé et a ainsi choisi la décomposition du problème et en a proposé une abstraction.

Le *co-pilote* développera pour sa part, la compétence E3. Une observation fine des activités des élèves pourra aussi mettre en évidence la maîtrise de la compétence G1 (écriture de tableaux en compréhension).

En résumé, sur l'ensemble de l'activité, ce sont principalement les compétences d'anticipation qui vous pouvoir être développées par le *pilote* et celles d'évaluation par le *co-pilote*. L'échange de rôles institutionnalisé par le *pair-programming*, est ainsi dans ce contexte un moyen d'amener les élèves à développer toutes les compétences.

On en déduit aussi qu'il n'y a pas lieu d'inscrire à notre référentiel de compétence spécifique au test. L'activité de test est une activité transversale qui gagne à être pratiquée tout au long des activités de programmation. Les compétences développées au cours des tests sont spécifiquement les compétences d'évaluation ainsi que celles de compréhension des modélisations fournies par les spécifications.

5 Expérimentation en classe de première NSI et premiers résultats

La mise au point de cette expérimentation a été effectuée au cours du premier trimestre 2023. La séquence a été testée en classe le 17 avril 2023 avec une classe de première NSI du lycée Le Verger à Sainte-Marie (La Réunion).

La classe comporte 18 élèves dont 16 garçons et 2 filles. La progression annuelle de cette classe a permis d'introduire les tableaux et les parcours simples au premier trimestre. Les activités 1 à 3 consistent donc en un réinvestissement de notions déjà abordées, alors que l'activité 4 a permis l'introduction au tri d'un tableau. Ce choix a été fait pour simplifier la genèse instrumentale avec un artefact nouveau, assez différent des notebooks avec lesquels les élèves avaient pris l'habitude de travailler en Python.

Le dispositif de recueil de données a permis de récolter les traces d'activités de 6 binômes. Chaque trace (en JSON) comporte l'horodatage de toutes les actions du *pilote* dans l'éditeur (charger, exécuter, partager) et du *co-pilote* dans sa console (charger, enregistrer). Par exemple, la figure 4 montre un extrait de la trace d'activité d'un *co-pilote,* au moment où il signale une erreur de programmation lors de l'activité 2 : « Si on lance le programme 2 fois avec le même tableau, ça fait une erreur », erreur provoquée par l'oubli d'initialisation de la variable indice.

```
2023-04-17T10:55:14.121000
MODE    : co-pilote
ACTION  : charger
EDITEUR :
def positionne_minimum(valeurs):
    mini=valeurs[0]
    for i in range(len(valeurs)):
        if mini>valeurs[i]:
            indice=i
            mini=valeurs[i]
    nb=valeurs[0]
    valeurs[0]=mini
    valeurs[indice]=nb
    return valeurs

2023-04-17T10:57:56.797000
MODE    : co-pilote
ACTION  : charger
EDITEUR :
def positionne_minimum(valeurs):
    mini=valeurs[0]
    indice=0
    for i in range(len(valeurs)):
        if mini>valeurs[i]:
            indice=i
            mini=valeurs[i]
    nb=valeurs[0]
    valeurs[0]=mini
    valeurs[indice]=nb
    return valeurs

2023-04-17T11:01:20.078000
MODE    : co-pilote
ACTION  : enregistrer
EDITEUR :
def positionne_minimum(valeurs):
    mini=valeurs[0]
    indice=0
    for i in range(len(valeurs)):
        if mini>valeurs[i]:
            indice=i
            mini=valeurs[i]
    nb=valeurs[0]
    valeurs[0]=mini
    valeurs[indice]=nb
    return valeurs
CONSOLE :
Brython 3.11.2
>>> valeurs = [40, 50, 10, 20, 30]
>>> positionne_minimum(valeurs)
[10, 50, 40, 20, 30]
>>> positionne_minimum(valeurs)
[10, 50, 40, 20, 30]
```

Figure 4 : Extrait de la trace d'activité du co-pilote du binôme 3.

On constate que les tests effectués après correction de la fonction permettent de vérifier que cette erreur a bien été corrigée.

Le tableau 2 résume les indicateurs calculés à partir de l'analyse des logs via un programme Python développé pour l'occasion. Nous avons en particulier calculé le nombre de partages de code par activité pour chaque binôme, et testé les programmes partagés avec des jeux de tests dédiés pour en déduire la réussite de chaque binôme à chaque activité.

Tableau 2: Indicateurs d'activité et de réussite des binômes par activité.

Binôme\Act	Activité 1 Partages	Activité 1 Réussite	Activité 2 Partages	Activité 2 Réussite	Activité 3 Partages	Activité 3 Réussite	Activité 4 Partages	Activité 4 Réussite
1	2	oui	1	oui	1	oui	1	oui
3	5	oui	3	oui	1	oui	1	oui
4	1	oui	2	oui	0	non	0	non
6	3	oui	3	oui	3	oui	2	oui
8	2	oui	1	oui	1	oui	1	oui
9	5	oui	3	oui	0	non	0	non

L'analyse globale des traces d'activités a ainsi permis de mettre en évidence les interactions entre *pilotes* et *co-pilotes* (attesté par le nombre de partages de code pour chaque activité), ainsi qu'une réussite globale à l'activité supérieure à ce qui pouvait être constaté habituellement par l'enseignante. Vu le faible nombre d'élèves, ces résultats n'ont bien sûr pas de valeur statistique, mais nous semblent cependant encourageants par rapport à l'intérêt de la méthode.

6 Conclusion

Ce travail en cours explore une méthodologie d'apprentissage de la programmation - le *pair-programming* - et propose un artefact prototype adapté en tant qu'objet d'étude. L'expérimentation a aussi permis d'améliorer la méthodologie de recueil de traces d'activités.

Il utilise par ailleurs le travail - en cours aussi - d'élaboration d'un référentiel de compétences en programmation adapté à la spécialité NSI.

Nous espérons ainsi contribuer à une meilleure compréhension des difficultés rencontrées par les élèves lors de leur apprentissage, en identifiant finement les compétences en jeu et en proposant des instruments incitant à diversifier les activités et amenant les élèves à collaborer.

Parmi les perspectives de cette étude exploratoire, nous envisageons d'une part une étude qualitative plus longue expérimentant de manière systématique le *pair-programming* sur une durée beaucoup plus longue dans une classe, et d'autre part une étude quantitative de l'appropriation de l'artefact avec un plus grand nombre d'élèves ou étudiants.

Références

Broisin, J., Venant, R., & Vidal, P. (2015). Lab4CE: a Remote Laboratory for Computer Education. *International Journal of Artificial Intelligence in Education*, *25*(4), 154-180.

Declercq, C. (2022). *Didactique de l'informatique : Une formation nécessaire*. Sticef *28*. http://sticef.org/num/vol2021/28.3.8.declercq/28.3.8.declercq.htm

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education : A literature review. *Computer Science Education*, *21*(2), 135-173. https://doi.org/10.1080/08993408.2011.579808

Schümmer, T., & Lukosch, S. G. (2009). Understanding Tools and Practices for Distributed Pair Programming. *J.UCS (Annual Print and CD-ROM Archive Ed.)*. https://www.narcis.nl/publication/RecordID/oai:tudelft.nl:uuid:9a29882c-8a6b-441b-b83b-e61d5a0b448b

Selby, C., & Woollard, J. (2014). Computational thinking : The developing definition. *SIGCSE*.

Tunga, Y., & Tokel, T. (2018). *The use of pair programming in education : A systematic literature review*.

Williams, L., & Kessler, R. (2003). *Pair programming illuminated*. Addison-Wesley.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*.