# Pensée informatique et activités de programmation : quels outils pour enseigner et évaluer ?

Kevin Sigayret, André Tricot, Nathalie Blanc

# Pensée informatique et activités de programmation : quels outils pour enseigner et évaluer ?

Kevin Sigayret, Nathalie Blanc, André Tricot
Université Montpellier 3, Laboratoire EPSYLON EA 4556, Site Saint Charles,
1 rue du Professeur Henri Serre, 34080 Montpellier
ksigayret@univ-Montp3.fr, nathalie.blanc@univ-montp.fr,
andre.tricot@univ-montp3.fr

## RÉSUMÉ

Au cours de cet atelier, nous présenterons une revue des outils d'évaluation de la pensée informatique et de leurs limites. Nous proposerons également deux nouveaux outils, que nous avons mis au point, et qui permettent de distinguer deux aspects fondamentaux de cette pensée informatique : la compréhension des notions et la capacité à résoudre des problèmes. Nous réaliserons ensuite une analyse comparative des trois principaux dispositifs d'enseignement de la programmation à l'école (activités débranchées, logiciels éducatifs et robotique). Nous verrons qu'il est aujourd'hui impossible de conclure sur l'efficacité relative de ces dispositifs pour développer la pensée informatique, par manque d'études fiables. Nous terminerons par un retour d'expérience relatif à des expérimentations que nous avons menées récemment et qui visaient à recueillir davantage de données empiriques concernant l'impact de ces dispositifs sur la capacité à acquérir la pensée informatique.

## ABSTRACT

**Computational thinking and programming activities :**
**which tools should be used to both teach and evaluate ?**
In this workshop, we present a review of tools used to assess computational thinking skills and discuss the limitations of these tools. We also present two new tools we developed, to enable the distinction between two fundamental aspects of computational thinking: conceptual understanding and problem solving skills. Then, we provide a comparative analysis of the three main devices used to teach programming at school (unplugged activities, educational software and robotics). Overall, we will highlight that it is impossible today to conclude on the relative effectiveness of these devices to develop computational thinking because of the lack of reliable studies. Finally, we draw some conclusions based on experimental works we recently conducted. In sum, we stress the need to gather more empirical data about the impact of these devices on the ability to acquire computational thinking.

MOTS-CLÉS : Programmation, apprentissage, pensée informatique, évaluation, activités débranchées, Scratch, robotique éducative

KEYWORDS: Programming, learning, computational thinking, assessment, unplugged activities, Scratch, educational robotics

*Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH), Fribourg, Suisse, 07 juin 2021, pages 68–75,* 

# 1 Introduction

Cette contribution s'inscrit dans le cadre d'une thèse réalisée au laboratoire de Psychologie Epsylon, rattaché à l'Université Paul Valéry Montpellier 3, sous la direction des Pr. Nathalie Blanc et André Tricot. La Délégation Académique au Numérique Éducatif (DANE) de l'Académie de Montpellier finance et suit de près ce travail de recherche. Cette thèse porte sur l'apprentissage de la programmation en contexte scolaire et sur l'acquisition de la pensée informatique.

En introduction, nous reviendrons sur la définition de la pensée informatique, telle que formulée à l'origine par Wing (2006). Nous verrons également qu'il n'existe pas véritablement de consensus sur la définition exacte de cette pensée informatique (Selby & Woollard (2013). D'où la difficulté qui en découle pour évaluer l'acquisition de cette pensée informatique et pour se prononcer sur l'efficacité des activités et dispositifs censés la développer.

Quelles sont les différentes approches existantes pour évaluer la maîtrise de la pensée informatique ? Quelles en sont les limites ?

L'apprentissage de la programmation à l'école permettrait de développer cette pensée informatique. Quels outils et dispositifs sont les plus indiqués pour enseigner cette discipline dans un cadre scolaire ?

Nous proposerons une revue de la littérature empirique relative à ces questions tout en apportant également notre contribution à ce débat. Deux nouveaux outils d'évaluation de la pensée informatique, actuellement en cours de validation psychométrique, seront proposés. Par ailleurs, un retour d'expérience sera réalisé concernant une étude expérimentale que nous avons menée au début de l'année 2021.

# 2 Quels outils pour évaluer la maîtrise de la pensée informatique ?

## 2.1 Revue des outils déjà existants

Dans cette première partie, nous reviendrons sur les différentes approches qui sont actuellement proposées pour évaluer l'acquisition et la maîtrise de la pensée informatique. Les approches qualitatives fondées sur des entretiens ou des observations peuvent présenter un intérêt certain. Cependant, nous nous focaliserons ici sur les outils permettant la récupération de données quantitatives, objectives et fiables, qui ont l’avantage de permettre des comparaisons.

Nous distinguerons notamment trois catégories d'outils qui correspondent à trois façons d'appréhender cette pensée informatique :

- Korkmaz et al. (2017) proposent le *Computational Thinking Scales* (CTS) qui regroupe des items issus de plusieurs tests distincts visant à évaluer différentes facultés cognitives. Cette approche considère la pensée informatique comme une agrégation de plusieurs compétences déjà bien identifiées (créativité, esprit critique, pensée logico-mathématique, compétences en résolution de problèmes...).

- D'autres chercheurs défendent une approche fondée sur l'analyse objective du code produit par un élève lors de la réalisation d'activités de programmation (Brennan & Resnick, 2012 ; Moreno-Léon & Robles, 2015 ; Alves et al., 2019). Ceux-ci estiment qu'il est possible d'associer l'utilisation de certains blocs issus des logiciels de programmation visuels à la maîtrise de certains concepts de la pensée informatique.

- Enfin, une dernière approche consiste à mettre en œuvre des tâches de résolution de problèmes dont l'accomplissement est censé traduire la maîtrise de certains aspects de la pensée informatique (Grover et al., 2014 ; Atmazidou & Demetriadis, 2016). Le *Computational Thinking Test* de Gonzalez (2015) représente probablement la tentative la plus aboutie en ce sens.

## 2.2 Deux nouveaux outils pour évaluer la pensée informatique

Globalement, les outils de mesure de la pensée informatique que nous avons détaillés précédemment sont des tentatives louables et pertinentes d'évaluer cette compétence. Cependant, ils ne proposent actuellement aucune distinction claire entre d'une part les connaissances des élèves et d'autre part leur capacité à appliquer ces connaissances pour résoudre des problèmes complexes. Or, nous considérons que cette distinction est capitale.

Tricot et Musial (2020) définissent la compétence comme un ensemble {Tâche ; Connaissances théoriquement nécessaires à la tâche ; Connaissances issues de la tâche}. En suivant ce modèle, la pensée informatique peut être considérée comme un ensemble regroupant une tâche (la résolution de problèmes algorithmiques), des connaissances théoriques nécessaires à la réalisation de cette tâche (les notions fondamentales en programmation) et des connaissances pratiques issues de cette tâche qui permettent au sujet d'utiliser les stratégies adéquates pour y faire face.

Si l'objectif est de développer une méthode d'évaluation permettant de discriminer différents niveaux de maîtrise de la pensée informatique, alors il faut pouvoir en distinguer les aspects les plus basiques, la compréhension des notions et concepts fondamentaux, et les aspects les plus complexes, l'assimilation et l'application pratique de ces concepts et le développement de stratégies permettant la résolution de problèmes.

Nous proposons ainsi deux nouveaux tests que nous avons conçus pour évaluer d'une part la compréhension des notions fondamentales en programmation et d'autre part la capacité à résoudre des problèmes algorithmiques. Le test de compréhension se compose de plusieurs questions à choix multiples. Le test d'algorithmique se présente sous la forme de problèmes à résoudre en produisant un algorithme, c'est-à-dire une série d'instructions qui constituent une solution à ce problème.

## 2.3 Validation psychométrique

Les deux tests ont suivi une procédure de validation psychométrique. Nous avons fait passer le test de compréhension des notions de programmation à près de 800 participants et le test d'algorithmique a été soumis à plus de 500 participants. Tous les participants étaient des élèves francophones de cycle 3 ou 4.

Chaque élève a dû, au préalable, remplir un questionnaire visant à estimer son niveau de pratique de la programmation. Deux versions de chaque test ont été créés : une version papier et une version numérique.

Les critères retenus pour s'assurer de la validité des tests sont sa sensibilité au niveau de pratique des élèves ainsi que la cohérence interne des items proposés. En d'autres termes, on s'attend à ce que le niveau de pratique de la programmation des participants soit positivement corrélé à leurs performances aux tests. On s'attend également à retrouver une corrélation entre les réponses aux items qui visent à mesurer le même construit (par exemple, la compréhension de la notion de boucle). Enfin, on s'assurera également de l'équivalence entre les versions papier et numérique des tests.

L'analyse des résultats obtenus concernant le test de compréhension des notions de programmation fait état d'une corrélation modérée entre le niveau de pratique des élèves et leurs performances au test ($r = 0.60$). Les indices de cohérence interne des items mesurant la compréhension des notions d'algorithme, de boucle, de condition ou de variable diffèrent suivant la version du test. La cohérence interne oscille entre $\alpha = 0.50$ et $\alpha = 0.54$ pour ces 4 notions dans la version numérique. Elle est plus élevée pour la version papier (entre $\alpha = 0.60$ et $\alpha = 0.66$). Les élèves de Cycle 4 présentent des niveaux de pratique significativement plus élevés que les élèves de Cycle 3, ce qui est cohérent et peut donc être considéré comme un signe de fiabilité du questionnaire visant à mesurer le niveau de pratique des élèves.

Pour le test d'algorithmique, les performances globales au test sont modérément corrélées au niveau de pratique de la programmation des élèves ($r = 0.58$). Malheureusement, peu d'élèves ont répondu à l'ensemble des exercices et seule une faible minorité est allé au-delà du 2ème exercice. Par conséquent, la seule mesure de la cohérence interne qui a pu être effectuée concerne la capacité des élèves à produire des algorithmes comprenant des instructions cohérentes et pertinentes et permettant de réaliser le parcours demandé dans les exercices 1 et 2 ($\alpha = 0.65$).

Les deux tests proposés présentent des résultats encourageants mais ne peuvent pas encore être considérés comme des outils suffisamment fiables et valides pour évaluer l'apprentissage de la programmation et la maîtrise de la pensée informatique. Pour le test de compréhension, la cohérence interne des items est actuellement insuffisante ($\alpha < 0.70$), même si la version papier est proche de cette valeur seuil. Cependant, des améliorations sont possibles pour accroître la fiabilité du test. Par exemples, les différents items présentent des niveaux de difficulté trop disparates et la possibilité de répondre correctement par hasard doit être diminuée, notamment en multipliant le nombre de réponses proposées.

En ce qui concerne le test d'algorithmique, des modifications vont être apportées pour diminuer la complexité des exercices et clarifier certaines formulations de phrases qui semblent n'avoir pas été parfaitement comprises par les élèves. De plus, lors de la passation des tests, les élèves ont souvent dû enchaîner les deux tests à la suite ce qui a rallongé la durée de la tâche et a probablement joué un rôle sur leur motivation. La passation des deux tests doit donc être réalisée séparément.

Une deuxième version de chaque test est en cours de réalisation afin de poursuivre le travail de validation psychométrique de ces outils.

# 3 Quels dispositifs pour enseigner la pensée informatique ?

## 3.1 Approche comparative des différents dispositifs utilisés

L'apprentissage de la programmation à l'école a pour finalité l'acquisition de connaissances et de compétences identifiées comme faisant partie de la pensée informatique (résolution de problèmes, aptitudes au raisonnement logique, créativité...). Il existe cependant diverses manières d'enseigner cette discipline. Nous proposerons ici une analyse comparative des trois principaux dispositifs utilisés :

- Les activités débranchées, qui ne font appel à aucun outil numérique, pourraient favoriser l'apprentissage en supprimant la charge cognitive liée à l'utilisation d'une machine tout en s'accordant avec les théories de la cognition incarnée qui soulignent l'importance des actions sensori-motrices concrètes dans le processus d'apprentissage (Romero et al., 2018)

- D'autres chercheurs ont mis en évidence l'intérêt de l'utilisation des logiciels visuels de programmation (Saez-Lopez et al., 2016 ; Xu et al., 2019). La richesse des possibilités offertes et le feedback immédiat renvoyé par l'ordinateur pourraient faciliter l'acquisition des compétences pratiques nécessaires pour résoudre des problèmes algorithmiques. Cependant, le coût cognitif de l'utilisation des logiciels pourrait également constituer une contrainte, diminuant les ressources attentionnelles disponibles et nécessaires à la compréhension.

- L'aspect tangible de la robotique éducative pourrait être un atout dans l'enseignement de la programmation tout en garantissant la grande diversité de possibilités et le feedback immédiat permis par les outils numériques (Sullivan & Heffernan, 2016 ; Scherer et al., 2020). Cependant, la recherche doit encore démontrer la validité empirique de cette supposition.

Nous conclurons cette analyse en précisant qu'à l'heure actuelle, il est impossible de se prononcer avec certitude sur l'efficacité relative de ces dispositifs comme outils permettant l'initiation à la programmation et le développement de la pensée informatique, en raison d'un manque de données empiriques fiables.

## 3.2 Présentation des expérimentations réalisées

Afin de faire progresser nos connaissances concernant l'efficacité relative des trois dispositifs cités précédemment, nous avons mené une expérience à laquelle ont pu participer plus de 350 élèves de cycle 3 issus de 14 classes de l'Académie de Montpellier. Ces élèves ont été répartis dans trois conditions expérimentales. Un premier groupe participait à des activités de programmation débranchées, sans utiliser aucun outil numérique. Le deuxième groupe était initié à la programmation via le logiciel éducatif Scratch. Enfin, le dernier groupe participait également à des activités sur Scratch mais, au lieu de programmer des personnages virtuels sur l'écran, le logiciel était connecté à un robot pédagogique Thymio que les élèves pouvaient ainsi apprendre à contrôler.

Les expérimentations ont eu lieu au cours des mois de Janvier et Février 2021. Au total, 10 séances de 45 minutes réparties sur 5 semaines ont été mises en place. Ces 10 séances ont été conçues pour être « équivalentes » entre les trois conditions.

Les notions abordées et les compétences travaillées étaient les mêmes dans les trois groupes expérimentaux, la seule différence reposait sur l'utilisation ou non de certains outils (logiciel et robot).

L'expérience s'est déroulée dans le cadre « naturel » de la classe afin de mettre en évidence des effets directement observables en pratique en contexte scolaire. Tous les élèves étaient novices en programmation au moment où les premières séances ont débuté.

## 3.3 Analyse des résultats obtenus

L'analyse des résultats est en cours de finalisation. L'objectif de cette expérience était de bénéficier de données quantitatives et objectives permettant la comparaison entre trois dispositifs d'enseignement différents afin de mettre en évidence leur efficacité relative dans l'apprentissage de la programmation à l'école. Pour ce faire, plusieurs variables ont été étudiées :

- La compréhension des notions fondamentales en programmation.
- La capacité à résoudre des problèmes algorithmiques.
- La motivation et le sentiment d'auto-efficacité.
- L'intérêt pour les disciplines scientifiques.

Pour les deux premiers facteurs, nous avons utilisé les deux tests que nous avons conçus précédemment.

Les deux autres facteurs concernent le vécu subjectif des élèves. A l'aide d'une comparaison pré-test/post-test, nous sommes en mesure d'estimer si l'un de ces dispositifs est susceptible d'accroître chez les élèves ayant participé à des activités de programmation la motivation, le sentiment d'auto-efficacité ou l'intérêt pour les sciences. Pour ce faire, nous avons sélectionné des items issus de tests existant ayant déjà été validés (Weinburgh & Steele, 2000 ; Marsh et al., 2006 ; Kind et al., 2007).

Pour l'ensemble des résultats à suivre, le coefficient d de Cohen est mentionné entre parenthèses pour indiquer la taille des effets mis en évidence.

Malheureusement, en raison de l'absence de trois des quatre enseignants de la condition « Scratch + Robot Thymio » lors de la dernière séance, les résultats des élèves au test de compréhension des notions et au test d'algorithmique n'ont pas pu être pris en compte dans cette condition.

Cependant, la comparaison entre les élèves en condition « débranchée » et les élèves en condition « Scratch » révèle un niveau de compréhension des notions largement plus élevé chez les élèves en condition « Scratch » ($d = 0.87$). Cet effet n'est pas visible sur la compréhension de la seule notion d'algorithme pour laquelle les différences observées ne sont pas significatives. En revanche, cet effet est assez fort en ce qui concerne les notions d'instruction ($d = 0.73$) et de boucle ($d = 0.67$) et encore plus marqué en ce qui concerne les notions de condition ($d = 0.81$) et de variable ($d = 0.81$).

Les filles présentent des scores de compréhension légèrement plus élevés que les garçons mais cet effet est faible ($d = 0.20$) et non significatif ($p = .08$).

De même, la capacité à résoudre des problèmes algorithmiques est nettement plus élevée dans la condition « Scratch » en comparaison avec la condition « débranchée » (d = 0.74). Cet effet n'est pas significatif sur les deux premiers exercices proposés qui sont plus simples et plus concrets mais très fort sur les deux derniers exercices proposés qui sont plus complexes et plus abstraits (d = 0.95).

Cet effet reste assez faible en ce qui concerne la capacité à produire des instructions conditionnelles (d = 0.35), moyen en ce qui concerne la capacité à produire un algorithme comprenant des instructions pertinentes et cohérentes (d = 0.61) ou la capacité à produire des boucles (d = 0.55), très fort en ce qui concerne la capacité à manipuler des variables (d = 0.95).

La capacité à résoudre des problèmes algorithmiques des élèves de CM2 est largement plus élevée que celle des élèves de CM1 (d = 0.90). Cet effet ne vient cependant pas remettre en cause la supériorité de la condition « Scratch » sur la condition « débranchée ». Les élèves de CM1, largement minoritaires, sont répartis assez équitablement dans les deux conditions.

Le niveau de compréhension des notions et la capacité à résoudre des problèmes algorithmiques sont moyennement corrélés (r = 0.40).

Au niveau du vécu subjectif des élèves, il n'y a pas d'évolution significative pré-test / post-test de l'intérêt pour les sciences dans les trois conditions étudiées.

On peut toutefois constater une légère baisse de la motivation post-test dans la condition « débranchée » par rapport au niveau de motivation évalué pré-test (d = 0.31), alors que motivation et sentiment d'auto-efficacité restent stables dans les deux autres conditions. Pré-test, les élèves dans la condition « Scratch + Robot » présentent un niveau de motivation plus élevé que dans les deux autres conditions (d = 0.72 par rapport à la condition « débranchée », d = 0.50 par rapport à la condition « Scratch »). Enfin, les garçons présentent des scores significativement plus élevés que les filles en ce qui concerne un item relatif à la volonté de faire de la programmation sur son temps libre (d = 0.40), présent dans le post-test seulement.

# Références


Alves, N. D. C., Von Wangenheim, C. G., & Hauck, J. C. (2019). Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. Informatics in Education, 18(1), 17.

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. Robotics and Autonomous Systems, 75, 661-670.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada (Vol. 1, p. 25).

González, M. R. (2015). Computational thinking test: Design guidelines and content validation. In Proceedings of EDULEARN15 conference (pp. 2436-2444).

Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 57-62).

Kind, P., Jones, K., & Barmby, P. (2007). Developing attitudes towards science measures. *International journal of science education*, *29*(7), 871-893.

Korkmaz, Ö., Cakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). Computers in Human Behavior, 72, 558-569.

Marsh, H. W., Hau, K. T., Artelt, C., Baumert, J., & Peschar, J. L. (2006). OECD's brief self-report measure of educational psychology's most useful affective constructs: Cross-cultural, psychometric comparisons across 25 countries. *International Journal of Testing*, *6*(4), 311-360

Moreno-León, J., & Robles, G. (2015, November). Dr. Scratch : A web tool to automatically evaluate Scratch projects. In Proceedings of the workshop in primary and secondary computing education (pp. 132-133).

Romero, M., Viéville, T., Duflot-Kremer, M., de Smet, C., & Belhassein, D. (2018, August). Analyse comparative d'une activité d'apprentissage de la programmation en mode branché et débranché.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. Computers & Education, 97, 129-141.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.

Sullivan, F. R., & Heffernan, J. (2016). Robotic construction kits as computational manipulatives for learning in the STEM disciplines. Journal of Research on Technology in Education, 48(2), 105-128.

Tricot, A., & Musial, M. (2020). Précis d'ingénierie pédagogique. De Boeck Supérieur.

Weinburgh, M. H., & Steele, D. (2000). The modified attitudes toward science inventory: Developing an instrument to be used with fifth grade urban students. *Journal of Women and Minorities in Science and Engineering*, *6*(1).

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. Computer Science Education, 29(2-3), 177-204.