L'apprentissage de la programmation : quels outils pour évaluer le développement de la pensée informatique à l'école ?

Kevin Sigayret, Nathalie Blanc, André Tricot

Dans Enfance 2022/4 (N° 4), pages 479 à 500
Éditions Presses Universitaires de France

ISSN 0013-7545
ISBN 9782130834779
DOI 10.3917/enf2.224.0479

L'apprentissage de la programmation : quels outils pour évaluer le développement de la pensée informatique à l'école ?

Kevin Sigayret[a], Nathalie Blanc[a] & André Tricot[a]

RÉSUMÉ

Malgré l'arrivée de la programmation informatique dans les cursus scolaires, il subsiste de nombreuses incertitudes sur les moyens mis en œuvre pour évaluer son apprentissage. L'une des finalités principales de l'apprentissage de la programmation serait la maîtrise de la pensée informatique, dont le développement constituerait un enjeu éducatif majeur pour les décennies à venir. Le présent article propose donc de passer en revue les outils d'évaluation des compétences en pensée informatique et leurs limites. Diverses approches sont discutées : échelles auto-évaluatives, outils d'analyse du code produit par l'élève, tâches de résolution de problèmes. L'importance de distinguer la compréhension des notions et la capacité à résoudre des problèmes dans la construction de ces outils est abordée. L'objectif de cet article est de fournir aux chercheurs comme aux enseignants une synthèse concernant les différentes approches disponibles pour évaluer le développement de la pensée informatique en contexte scolaire. Cette synthèse aura des retombées sur les recherches à venir consacrées à l'évaluation de la pensée informatique et pourra alimenter la réflexion engagée sur les pratiques à l'école.

MOTS-CLÉS : PROGRAMMATION, APPRENTISSAGE, PENSÉE INFORMATIQUE, OUTILS D'ÉVALUATION.

ABSTRACT

Programming learning: what tools to evaluate computational thinking development in school?

Despite the introduction of computer programming in school curricula, there are still many uncertainties about the means used to assess its learning. One of the main purposes of learning programming is to master computational thinking, whose development would constitute a major educational challenge for the decades to come. This article there-

[a] Laboratoire EPSYLON EA 4556, 1 rue du Professeur Henri Serre, 34090 Montpellier, Université Paul Valéry Montpellier 3. *E-mails* : kevin.sigayret@univ-monpt3.fr ; nathalie.blanc@univ-montp3.fr ; andre.tricot@univ-montp3.fr.

fore proposes to review the tools for assessing computational thinking skills and their limitations. Various approaches are discussed: self-assessment scales, tools for analyzing the code produced by the student, problem-solving tasks. The importance of distinguishing between conceptual understanding and problem-solving skills in the construction of these tools is addressed. The aim of this article is to provide researchers and teachers with a synthesis of the different approaches available to assess the development of computational thinking in a school context.

KEYWORDS: PROGRAMMING, LEARNING, COMPUTATIONAL THINKING, ASSESSMENT TOOLS.

INTRODUCTION

La programmation prend incontestablement une importance de plus en plus grande dans notre société où la maîtrise du numérique et de ses possibilités n'est plus une option. Les premières tentatives d'introduction de la programmation dans les classes françaises datent des années 1960, avant que le Plan informatique pour tous de 1985 ne vienne plutôt centrer l'apprentissage sur l'utilisation de l'ordinateur lui-même. La diffusion rapide du web et des outils de communication à l'échelle mondiale à partir du début des années 2000 marque l'avènement d'une nouvelle période dans laquelle la prise en compte de la programmation en tant que discipline scolaire à part entière s'accélère.

De nos jours, selon le rapport Eurydice (Bourgeois, Birch & Davydovskaia, 2019), qui donne un aperçu de l'état de l'éducation numérique en Europe, les compétences en programmation et en codage sont des objectifs d'apprentissage pour l'enseignement primaire dans une vingtaine de systèmes éducatifs européens et dans une trentaine de pays en ce qui concerne l'enseignement secondaire. En France, la compétence numérique est scindée en deux axes dont le premier la définit comme une langue : les langages de programmation et les algorithmes. Le rapport Eurydice classe la programmation comme une compétence issue d'un domaine plus large qui est celui de la création de contenus numériques.

L'apprentissage de la programmation à l'école connaît un essor indéniable depuis vingt ans. Ceci peut s'expliquer en partie par le développement des logiciels de programmation visuelle (par exemple Scratch), qui rendent plus accessible cette discipline parfois jugée trop complexe, ainsi que par la nécessité pour le futur citoyen de pouvoir appréhender et démystifier le numérique et ses enjeux par une réflexion critique (Romero, Viéville, Duflot-Kremer, de Smet & Belhassein, 2018).

Au-delà de la nécessité de faire entrer la programmation dans un contexte scolaire pour former le citoyen de demain, il semblerait que son apprentissage stimule certaines facultés cognitives et permette d'accompagner le développe-

ment intellectuel des enfants et adolescents. Apprendre à programmer pourrait ainsi améliorer les aptitudes au raisonnement logique (Psycharis & Kallia, 2017) et avoir une influence positive sur certaines compétences dites de « haut niveau » (Popat & Starkey, 2019). De plus, Scherer, Siddiq et Sánchez Viveros (2019) ont mis en évidence la possibilité de transférer ce qui a été appris en programmation à d'autres situations, dans un contexte plus ou moins différent.

Alors que la programmation prend de plus en plus de place dans les programmes scolaires, les finalités de cette discipline et les moyens disponibles pour évaluer son apprentissage sont l'objet de discussions et de controverses. Il est aujourd'hui essentiel de permettre aux enseignants, qui ne sont pas toujours suffisamment formés pour faire face à ce défi, de bénéficier d'outils d'évaluation dont les apports et les limites ont été établis de manière rigoureuse.

Dans un premier temps, nous tenterons de définir précisément ce que l'on appelle le développement de la « pensée informatique » qui constitue l'un des enjeux majeurs de l'enseignement de la programmation à l'école. Wing (2006) estime ainsi que la pensée informatique est une compétence fondamentale et devrait faire partie des capacités analytiques de tous les enfants, au même titre que la lecture, l'écriture ou l'arithmétique. La nécessité pour tous les individus de maîtriser cette pensée informatique d'ici le milieu du XXI[e] siècle est un constat partagé par de nombreux chercheurs et experts du domaine (Mohaghegh & McCauley, 2016 ; Ioannou & Makridou, 2018). De même, Nogry (2018) estime que la transmission d'une culture informatique est devenue un enjeu sociétal. L'initiation à la pensée informatique permettrait ainsi au futur cybercitoyen d'acquérir une certaine maîtrise et donc de dépasser le statut de simple consommateur du numérique.

En français, « pensée informatique » est une traduction imparfaite de l'expression anglaise originale « computational thinking ». Cette dernière comprend le mot « thinking » qui désigne davantage le « processus de penser » que le produit de ce processus, qui est la pensée elle-même. Même si les termes « pensée computationnelle » ou « processus de pensée informatique » seraient plus appropriés, il semble que l'expression « pensée informatique » se soit imposée comme la traduction la plus répandue de « computational thinking ». C'est ainsi que nous désignerons la pensée computationnelle/les processus de pensée informatiques tout au long de cet article.

Cet article présente une revue et une synthèse des outils d'évaluation de la pensée informatique les plus robustes disponibles à ce jour dans la littérature. Les apports et les limites de ces outils sont discutés. L'objectif est de permettre aux chercheurs et aux enseignants d'accroître leurs capacités à évaluer les apprentissages de leurs élèves. Bien que les différentes méthodes, approches ou tests soient généralement décrits en anglais dans les études citées et aient été testés dans des pays étrangers, il est possible et souhaitable que les enseignants s'en inspirent afin de bénéficier d'outils d'évaluation dont la validité a été établie de manière rigoureuse. D'où l'intérêt de cette contribution qui a pour but de faire connaître ces outils et de les rendre accessibles aux professionnels de l'éducation.

Le retour de la programmation en tant que discipline scolaire et la généralisation de l'intérêt porté à la pensée informatique sont deux phénomènes assez récents. Par conséquent, les connaissances actuelles sont encore limitées et ne font pas l'objet d'un véritable consensus scientifique. Il est donc essentiel d'effectuer un tour d'horizon des différentes approches disponibles pour évaluer les apprentissages dans ce domaine. Les outils présentés ont été utilisés et validés auprès d'élèves de l'école élémentaire à la fin du secondaire et comportent des échelles auto-évaluatives, des outils d'analyse des travaux des élèves (focalisés sur le résultat produit ou sur les processus ayant mené à ce résultat) ainsi que des tâches de résolution de problèmes. Nous proposerons également une nouvelle approche qui nous semble pertinente pour évaluer les apprentissages en programmation.

Apprendre à programmer pour développer une pensée informatique

Apprendre à programmer

Le terme « programmation » regroupe généralement un ensemble d'activités relativement disparates qui convergent vers un but identique : développer la capacité de l'élève à produire des algorithmes, c'est-à-dire des successions d'actions qui constituent autant d'étapes vers la résolution d'un problème algorithmique. La programmation est considérée depuis ses débuts comme un processus très complexe impliquant un grand nombre d'activités cognitives et de représentations mentales (Rogalski & Samurçay, 1990). Généralement, les activités de programmation nécessitent l'analyse d'un problème donné afin de déterminer un algorithme adéquat qu'il faudra par la suite retranscrire dans un langage de programmation (Rogalski, Samurçay & Hoc, 1988).

D'un point de vue psychologique, programmer c'est élaborer une conception fonctionnelle de la relation entre la solution d'un problème et la représentation opératoire de cette solution dans un langage de programmation (Samurçay & Rouchier, 1985). Cette nécessaire prise en compte du dispositif d'exécution d'un programme distingue la programmation des activités de résolution de problèmes classiques et induit une modification de la planification des actions. Samurçay et Rouchier (1985) ont soumis des lycéens (âgés de 15-16 ans) à un problème mathématique simple dont la solution était connue de tous et ont montré qu'un grand nombre d'entre eux ne sont pas parvenus à programmer cette solution en prenant en compte les limitations de la machine qui leur étaient imposées. Les auteurs en ont conclu que les élèves avaient des difficultés à se mettre spontanément dans une situation de production de procédures plutôt que dans une situation de production de résultats.

Dans une revue de la littérature, Robins, Rountree et Rountree (2003) distinguent deux composantes de l'apprentissage de la programmation qui sont d'un côté les connaissances et de l'autre les stratégies employées pour program-

mer et résoudre un problème. La question est alors de déterminer pourquoi et comment différentes stratégies peuvent émerger, et comment celles-ci sont liées aux connaissances sous-jacentes. De même, ils différencient la compréhension d'un programme et la capacité à en générer. Pour les auteurs, il est clair que ces deux derniers aspects sont liés, la compréhension jouant un rôle important dans la création d'un programme, mais ils suggèrent par ailleurs que les capacités des individus en ce qui concerne ces deux tâches ne sont pas forcément bien corrélées. Les tâches d'évaluation basées sur la mesure de la compréhension d'un programme ne sont pas nécessairement de bons indicateurs de la capacité à concevoir et rédiger des programmes.

La programmation demeure indissociable de la notion d'algorithme bien que cette notion ne soit pas propre à l'informatique et à la programmation. Dans un document rédigé à l'attention des enseignants, Tchounikine (2017) définit l'algorithme comme « un enchaînement mécanique d'actions, dans un certain ordre, qui chacune a un effet, et dont l'exécution complète permet de résoudre ou de faire quelque chose ». Aujourd'hui, programmer et coder sont parfois considérés à tort comme synonymes. Le codage ne constitue cependant qu'une des phases du processus de programmation, celle où l'on écrit le programme. Ce processus inclut également d'autres phases, notamment celle qui consiste à concevoir et développer des algorithmes (Lodi & Martini, 2021) qui correspond davantage à ce que l'on cherche à transmettre dans l'enseignement scolaire. Le « socle commun de connaissances, de compétences et de culture » (Ministère de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 2015) indique notamment que l'élève « connaît les principes de base de l'algorithmique et de la conception des programmes informatiques. Il les met en œuvre pour créer des applications simples » (p. 4). Ce socle mentionne également les « langages informatiques [qui] sont utilisés pour programmer des outils numériques et réaliser des traitements automatiques de données (p. 4) ».

L'enseignement de la programmation à l'école s'appuie par ailleurs sur différents dispositifs censés faciliter la transmission des connaissances et compétences fondamentales dans cette discipline auprès d'un public jeune, en particulier les élèves d'école primaire. Les robots pédagogiques et les interfaces visuels basés sur des blocs à glisser-déposer (*drag & drop*) constituent l'essentiel des dispositifs qui sont utilisés aujourd'hui pour rendre la programmation accessible et attrayante (Bocconi, Chioccariello, Dettori, Ferrari & Engelhardt, 2016). Plusieurs auteurs ont également mis en évidence l'intérêt des activités débranchées comme moyen d'enseigner la programmation en éliminant complètement la charge cognitive liée à l'utilisation des machines et des outils numériques (Brackmann, Román-González, Robles, Moreno-León, Casali & Barone, 2017 ; Romero et al., 2018). Malgré plusieurs tentatives récentes visant à comparer l'effet de ces différents dispositifs sur les performances d'apprentissage des élèves (del Olmo-Muñoz, Cózar-Gutiérrez & González-Calero, 2020 ; Sigayret, Tricot & Blanc, 2022), beaucoup reste à faire pour identifier les apports et limites de ces outils.

Avant même de pouvoir envisager la possibilité d'évaluer l'apprentissage de la programmation, il est nécessaire de s'interroger sur les connaissances ou les compétences que cet apprentissage est censé développer chez l'apprenant. De nombreuses études ont tenté de mesurer l'impact des activités de programmation sur un large panel de facultés cognitives très générales. Généralement, l'ensemble des compétences cognitives spécifiques et des processus de résolution de problèmes qui interviennent notamment dans les activités de programmation sont regroupés sous le terme de « pensée informatique » qui désigne une capacité bien au-delà de la simple capacité à programmer. La pensée informatique peut plutôt être considérée comme l'ensemble des capacités cognitives de résolution de problèmes qui déterminent (entre autres) les compétences en programmation (Denning, 2017).

Dans une revue de la littérature relative à l'apprentissage de la pensée informatique à travers les activités de programmation, Lye et Koh (2014) affirment que la programmation est plus qu'une simple activité de codage, justement parce qu'elle implique que les apprenants manifestent une certaine maîtrise de la pensée informatique. Cependant, si programmation et pensée informatique sont résolument liées, il est nécessaire de préciser que la programmation peut également amener à la construction de savoirs distincts de cette pensée informatique, par exemple en relation avec la maîtrise technique des outils numériques utilisés. De même, la pensée informatique ne se développe pas exclusivement par le biais des activités de programmation et demeure une attitude et une compétence universellement applicable que tout le monde, pas seulement les informaticiens, devrait apprendre et utiliser (Wing, 2006).

Développer une pensée informatique

Roman-González, Pérez-González et Jiménez-Fernández (2017) distinguent trois types de définitions de la pensée informatique :

les définitions générales plutôt focalisées sur l'ensemble des aptitudes impliquées dans la résolution de problèmes (Wing, 2006 ; Wing, 2011 ; Aho, 2012) ;
les définitions opérationnelles qui fournissent un cadre en catégorisant la pensée informatique en différents sous-domaines ou processus cognitifs (CSTA & ISTE, 2011 ; Selby & Woollard, 2013) ;
les définitions pédagogiques qui listent un ensemble de concepts et de compétences (savoirs et savoir-faire) à enseigner dans un contexte éducatif (Brennan & Resnick, 2012 ; Zhang & Nouri, 2019).

Wing (2006) décrit la pensée informatique, ou pensée computationnelle (*computational thinking*), comme une manière de résoudre des problèmes ou de concevoir des systèmes qui s'appuie sur des concepts fondamentaux de la science informatique. Cette pensée informatique partage de nombreuses similarités avec la programmation, les mathématiques, l'ingénierie et, plus globalement, la pensée scientifique. C'est plus précisément « l'ensemble des processus de pensée impliqués dans la formulation d'un problème et l'expression de sa

solution de manière à ce qu'un ordinateur - humain ou machine - puisse effectivement l'exécuter » (Wing, 2011) ou de manière à ce que « sa solution puisse être représentée sous formes d'étapes de traitement ou de calcul (« *computational steps* ») et d'algorithmes » (Aho, 2012, p. 1).

Bien des années plus tôt, Papert (1980) fut le premier à utiliser l'expression « pensée informatique » en faisant référence à une aptitude mentale que les enfants développent en programmant avec le langage Logo. Il suggérait notamment que l'informatique pouvait offrir aux enfants de nouvelles possibilités d'apprentissage, de réflexion, de développement émotionnel et cognitif et favoriser la pensée procédurale par le biais des activités de programmation.

La définition exacte de la pensée informatique et de ce qu'elle implique demeure cependant sujette à interprétations et il n'existe pas de véritable consensus sur sa description. Selby et Woollard (2013) ont tenté d'en apporter une définition plus précise afin notamment de faciliter l'élaboration d'outils permettant d'évaluer les compétences en pensée informatique, dans un cadre scolaire par exemple. Pour ce faire, les auteurs ont compilé et analysé les articles relatifs à cette pensée informatique, dans l'optique d'en dégager les éléments récurrents qui pourraient contribuer à la définir de manière plus rigoureuse. Il en résulte que trois expressions ou termes apparaissent systématiquement dans la littérature. La pensée informatique semble ainsi inclure inévitablement la notion de « processus de pensée » ainsi que les termes « abstraction » (prise en compte simultanée de plusieurs niveaux d'abstraction) et « décomposition » (capacité à décomposer un problème complexe en plusieurs problèmes plus simples).

D'autres termes sont aussi largement utilisés mais de manière moins consensuelle. C'est le cas d'un ensemble de termes relatifs à des formes de pensée (« pensée logique », « pensée mathématique », « pensée heuristique », etc.) ou aux compétences de « résolution de problèmes ». En particulier, le terme « généralisation », qui désigne la capacité à transférer la résolution d'un problème particulier à tout un ensemble de problèmes plus généraux, et le terme « évaluation », qui correspond à la capacité à reconnaître et évaluer des résultats, sont des termes très souvent usités.

Selby et Woollard (2013) proposent une définition qui inclut ces mots qui sont les plus fréquemment utilisés. La pensée informatique est ainsi définie comme une activité, souvent orientée vers la production de quelque chose, associée (mais pas limitée) à la résolution de problèmes. Il s'agit, plus globalement, d'un processus cognitif qui reflète la capacité à penser en termes d'abstraction, de décomposition, d'évaluation, de généralisation et de manière algorithmique (capacité à écrire des instructions spécifiques et explicites pour chaque étape d'un processus). Cependant, ces processus identifiés par Selby et Woollard (2013) comme composantes principales de la pensée informatique ne sont pas nécessairement spécifiques à ce domaine et peuvent être retrouvés dans un grand nombre d'activités cognitives (liées à la lecture ou aux mathématiques par exemple). Ceci traduit plus généralement la difficulté actuelle des

chercheurs à définir la pensée informatique en des termes suffisamment précis et spécifiques.

La *Computer Science Teachers Association* et *l'International Society for Technology in Education* ont également tenté d'apporter une définition opérationnelle de la pensée informatique (CSTA & ISTE, 2011). La pensée informatique y est ainsi décrite comme un « processus de résolution de problèmes qui comprend les caractéristiques suivantes : formuler les problèmes d'une manière qui permette d'utiliser un ordinateur et d'autres outils pour aider à les résoudre ; organiser logiquement et analyser des données ; représenter des données par des abstractions telles que des modèles et des simulations ; automatiser des solutions à travers la pensée algorithmique ; identifier, analyser et mettre en œuvre des solutions possibles dans le but d'obtenir la combinaison la plus efficiente et efficace d'étapes et de ressources ; généraliser et transférer ce processus de résolution de problèmes à une grande variété de problèmes ». Plus tard, l'ISTE a également affirmé que ce sont d'autres compétences bien connues (la créativité, la pensée algorithmique, la pensée critique, la résolution de problèmes, la pensée coopérative et les compétences communicationnelles) qui, considérées conjointement, constituent cette nouvelle compétence que l'on nomme pensée informatique (ISTE, 2015, citée dans Korkmaz, Cakir & Özden, 2017).

On peut cependant remarquer que les définitions opérationnelles présentées ci-dessus ne sont pas spécifiques à l'apprentissage de la programmation. *A contrario*, Brennan et Resnick (2012) établissent un cadre théorique, comprenant 3 dimensions essentielles, qui permet de définir et d'enseigner la pensée informatique à l'école en s'appuyant sur le logiciel Scratch. Ils y distinguent :

– les concepts computationnels qui regroupent les notions à connaître en programmation : séquences (série d'étapes ou d'instructions), événements (action en provoquant une autre), boucles (répétition d'une partie du programme), opérateurs (expression mathématique ou logique), parallélisme (séquences d'instructions se produisant en même temps), conditions (exécution d'une partie du programme uniquement lorsqu'une certaine condition est remplie), données (stocker, retrouver et actualiser des valeurs).

– les pratiques computationnelles : être incrémental et itératif (planifier la conception d'un programme avant de l'implémenter), tester et débugger (développer des stratégies pour faire face aux problèmes et les anticiper), réutiliser et remixer (s'appuyer sur des programmes antérieurs ou réalisés par d'autres), abstraire et modulariser (construire un programme complexe en assemblant des parts plus petites).

– les perspectives computationnelles : s'exprimer, se connecter, questionner.

Dans leur revue systématique de la littérature consacrée à l'enseignement de la pensée informatique via Scratch, Zhang et Nouri (2019) identifient un ensemble de compétences qui complète les travaux de Brennan et Resnick (2012) : la capacité à lire, interpréter et communiquer du code ; la pensée prédictive (prédire l'exécution d'un programme avant de le concevoir), les notions d'entrées et de sorties, l'utilisation de supports multimodaux (capacité

à synchroniser des images, des sons, des actions…) ou encore les interactions hommes-machines (interactivité du programme créé).

D'autres modèles ont émergé pour tenter d'établir un cadre conceptuel solide permettant de définir précisément ce qui compose la pensée informatique (Grover & Pea, 2013 ; Kalelioglu, Gülbahar, & Kukul, 2016 ; Shute, Sun & Asbell-Clarke, 2017 ; Weintrop et al., 2016). Ces modèles recoupent partiellement ou totalement un ou plusieurs des modèles présentés précédemment. Toutefois, la multiplication de ces propositions théoriques, bien que souvent proches les unes des autres, traduit l'absence de consensus de la communauté sur cette question.

Il semblerait cependant que l'apprentissage de la programmation et de la pensée informatique soient inévitablement rattachés à la notion de compétence (« *Computational thinking skills* »), de façon plus ou moins explicite. Il est donc nécessaire de se demander ce qu'est une compétence et comment elle peut être évaluée. Van Lint (2016) analyse les définitions institutionnelles de la compétence issues de plusieurs pays européens pour mettre en évidence les composantes de cette notion. Les définitions sont globalement assez similaires. En France, la compétence se définit comme « l'aptitude à mobiliser des ressources (connaissances, capacités, attitudes) pour accomplir une tâche ou faire face à une situation complexe ou inédite » (Ministère de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 2015).

Selon Van Lint (2016), la compétence se distingue ainsi de la connaissance qui regroupe un ensemble de concepts, d'énoncés, de définitions, de règles, de formules ou de faits à comprendre et à mémoriser, et de la procédure qui désigne un enchaînement organisé d'actions automatisées qui restent les mêmes dans toutes les situations. La compétence est plutôt une aptitude qui se révèle dans des situations complexes nouvelles qui nécessitent une certaine adaptation. Cette aptitude repose sur la mise en œuvre de connaissances et de procédures adéquates dans une situation donnée (Van Lint, 2016). Pour évaluer une compétence, il convient donc de placer l'élève face à une situation nouvelle et complexe (c'est-à-dire mobilisant plusieurs ressources) qui exige, de la part de l'apprenant, l'accomplissement d'une tâche.

En ce qui concerne l'apprentissage de la programmation dans l'enseignement scolaire français, Tchounikine (2017) identifie 5 notions fondamentales qui constituent les bases que les élèves doivent connaître et comprendre : la notion d'algorithme (ou de programme), la notion d'instruction (ou d'action), la notion de condition (« Si… alors… »), la notion de boucle et, éventuellement, la notion de variable. En outre, l'auteur présente les compétences sous-jacentes à la pensée informatique de la manière suivante : « savoir décomposer un problème en sous-problèmes plus simples ; savoir réfléchir aux tâches à accomplir pour résoudre un problème en termes d'étapes et d'actions ; savoir décrire les problèmes et les solutions à différents niveaux d'abstraction, ce qui permet d'identifier des similitudes entre problèmes et, par la suite, de pouvoir réutiliser des éléments de solution ». On retrouve ici des compétences similaires à celles décrites notamment par Selby et Woollard (2013).

D'une manière plus générale, certains chercheurs considèrent la pensée informatique comme un ensemble de compétences, exigeant des apprenants qu'ils développent à la fois des connaissances spécifiques à un domaine (par exemple, la programmation) et des aptitudes à la résolution de problèmes (Tang, Yin, Lin, Hadad & Zhai, 2020). Ceux-ci suggèrent que ces compétences pourraient être utilisées pour résoudre des problèmes de la vie quotidienne, dans différents domaines et à tous les niveaux scolaires.

Dans la suite de cet article, nous retenons que la pensée informatique peut regrouper : (a) la maîtrise de concepts informatiques : algorithme, instruction, séquence et parallélisme, événement, opérateurs, données, condition, boucle et variable et (b) la compétence à résoudre certains problèmes nouveaux à partir d'abstraction, de décomposition, d'évaluation, de généralisation et de manière algorithmique mais aussi (c) la capacité à lire, interpréter et communiquer du code.

Quels outils pour mesurer l'acquisition de la pensée informatique ?

La question de l'évaluation des compétences en pensée informatique est sujette à débat actuellement. La définition même de la pensée informatique et les compétences qu'elle met en jeu ne font pas consensus. Plusieurs outils sont proposés pour évaluer l'acquisition de la pensée informatique à l'école. De nombreux chercheurs recommandent d'ailleurs de pallier les difficultés rencontrées pour évaluer la pensée informatique en associant plusieurs outils complémentaires pour permettre d'avoir une idée plus précise du niveau des élèves (Roman-González, Moreno-León et Robles, 2019).

La méthode utilisée pour effectuer notre revue systématique de la littérature est décrite ci-dessous.

1. Nous avons utilisé la base de données transdisciplinaire Google Scholar. Le Tableau 1 présente les mots-clés qui ont été utilisés pour notre recherche ainsi que le nombre d'articles retenus. Des milliers d'articles abordent l'évaluation de la pensée informatique, ne serait-ce que dans leur partie théorique, sans pour autant être pertinents pour notre revue de littérature centrée sur les outils d'évaluation. Notre recherche était donc basée uniquement sur le titre des articles, ce qui nous a permis d'effectuer un premier filtrage en excluant tous ceux qui n'étaient pas véritablement centrés sur une présentation détaillée de ces outils d'évaluation. Les mots-clés sont présentés ici en anglais mais leurs équivalents français ont également été utilisés pour mener à bien cette sélection. Très peu d'études comprenant ces mots-clés dans leurs titres ont été publiées en français et aucune ne correspondait à nos critères d'inclusion. Cette recherche a été effectuée pour la dernière fois en décembre 2021.

2. Les références de chaque résultat obtenu à la suite de cette recherche ont été récupérées.

3. Après avoir lu le titre et le résumé, nous avons sélectionné 20 articles respectant les critères suivants :

Critères d'inclusion : seuls les articles, rédigés en français ou en anglais, centrés sur un outil, une méthode ou une approche visant à évaluer la maîtrise de la pensée informatique ont été sélectionnés. Nous avons également veillé à ce que les articles en question présentent une description précise de ces outils et/ou des données psychométriques attestant de leur validité et de leur fiabilité.

Critères d'exclusion : les doublons ont été exclus de notre sélection ainsi que les travaux non aboutis (études préliminaires, projets…) et les tests uniquement à destination d'une population adulte. Les approches qualitatives fondées sur des entretiens ou des observations ont également été exclues. Même si celles-ci peuvent présenter un intérêt certain, nous nous focalisons ici sur les outils d'évaluation permettant la récupération rapide de données quantitatives, objectives et fiables, qui ont l'avantage de faciliter la mesure de différents niveaux de performances et de rendre possible les comparaisons.

4. Les 20 articles ont par la suite été lus dans leur intégralité et analysés, ce qui nous a permis de découvrir 2 nouveaux articles qui apportent une contribution majeure et qui ont été intégrés à une deuxième sélection élargie et définitive (voir Tableau 1). Au total, 22 articles ont été retenus pour les besoins de cette revue de littérature.

Tableau 1. Sélection des articles en fonction des mots-clés utilisés.

Mots-clés	Résultats de la recherche	Articles sélectionnés	Total 1re sélection	Ajout d'articles	TOTAL
Computational Thinking Scale(s)	15	3	20	2	22
Computational Thinking Test	26	4			
Computational Thinking Assess	17	2			
Computational Thinking Assessing	61	4			
Computational Thinking Assessment	100	7			

Des échelles auto-évaluatives

Korkmaz et al. (2017) proposent les *Computational Thinking Scales* (CTS) qui regroupent des items issus de plusieurs tests distincts. Les auteurs s'appuient sur un document de l'ISTE (ISTE, 2015, citée dans Korkmaz et al., 2017) qui définit la pensée informatique comme un ensemble de compétences. Le CTS vise ainsi à mesurer les capacités créatives, la pensée algorithmique, la pensée critique, la capacité à résoudre des problèmes et les capacités coopératives. Tous les items qui les composent ont été récupérés à partir de tests préexistants

(*Problem Solving Scale, Cooperative Learning Attitude Scale, Scale of California Critical Thinking tendency*, etc.).

Le questionnaire est auto-administré, ce qui pose des problèmes liés à la subjectivité des réponses fournies par les élèves, et a été validé auprès de plus de 700 étudiants à l'université qui ont répondu aux 74 affirmations du test à l'aide d'une échelle de Likert en 5 points (jamais, rarement, parfois, souvent, toujours). Le test a donc suivi un processus rigoureux de validation mais la question de sa généralisation à des niveaux inférieurs, et donc de sa présence dans notre sélection, peut légitimement être posée. Pour autant, cette échelle a été testée avec succès auprès d'un millier d'étudiants chinois âgés de 15-16 ans (Korkmaz & Xuemei, 2019) ce qui nous laisse envisager la possibilité de pouvoir l'utiliser également en France à la fin de l'enseignement secondaire.

Cet outil d'évaluation est intéressant mais la pensée informatique est ici considérée comme la simple agrégation d'un ensemble de compétences bien connues et bien identifiées. S'il semble pertinent d'affirmer que la pensée créative ou la pensée critique sont liées à la pensée informatique, la proposition de Korkmaz *et al.* (2017) de considérer la seconde comme la combinaison des deux premières ainsi que d'autres facultés est aussi potentiellement critiquable car relativement réductrice. D'autres échelles du même type, visant à évaluer les perceptions et l'attitude des élèves vis-à-vis de la pensée informatique, ont été validées récemment, en particulier en ce qui concerne le sentiment d'auto-efficacité des élèves en relation avec la maîtrise de la pensée informatique (Weese & Feldhausen, 2017 ; Kukul & Karataş, 2019). Ces échelles présentent les mêmes limites que celle de Korkmaz et al. (2017) et s'adressent à un public plus jeune (plutôt les 10-14 ans). Globalement, les échelles auto-évaluatives peuvent être préconisées pour établir un autodiagnostic, faisant suite à une intervention ayant eu pour objectif de développer la pensée informatique.

Des outils d'analyse objectifs du code produit par l'élève

Brennan et Resnick (2012) proposent différentes façons d'évaluer la pensée informatique. Premièrement, en associant certains blocs du logiciel de programmation Scratch à certains concepts de la pensée informatique, il serait possible d'analyser de manière objective les projets Scratch réalisés par les élèves. Cependant, les chercheurs reconnaissent eux-mêmes que cette manière de procéder est limitée. L'utilisation de certaines instructions ne garantit pas que celles-ci soient parfaitement comprises et que les compétences associées soient bien maîtrisées.

De même, Alves, Von Wangenheim et Hauck (2019) font état de différents outils qui se fondent exclusivement sur l'analyse du code produit pour analyser les programmes réalisés sur des logiciels de programmation visuels, en particulier sur Scratch. Globalement, ce type d'évaluation est critiquable sur au moins un aspect majeur. Il admet que les compétences en pensée informatique sont visibles directement à travers l'analyse du code qui est généré par l'élève, ce qui

reste une supposition. Par ailleurs, cette approche limite l'évaluation de la maîtrise de la pensée informatique aux activités de programmation.

Dans ce registre, Dr. Scratch est un outil capable d'analyser automatiquement des fichiers Scratch, notamment pour évaluer différents aspects de la pensée informatique (Moreno-León & Robles, 2015). Un feedback est ensuite renvoyé à l'utilisateur avec trois niveaux de compétence (basique, développé, maîtrisé). Les auteurs ont réussi à montrer que ce feedback peut être bénéfique pour améliorer les performances de codage des élèves, et donc potentiellement la maîtrise sous-jacente des compétences liées à la pensée informatique (Moreno-León & Robles, 2015). Il existe par ailleurs de fortes corrélations entre les évaluations réalisées automatiquement via Dr. Scratch et les évaluations réalisées par des experts humains (Moreno-León, Román-González, Harteveld & Robles, 2017).

L'abstraction, la synchronisation, la représentation des données, l'interactivité, la pensée logique, le contrôle des flux (lié ici à l'utilisation pertinente de boucles « répéter ») et le parallélisme sont les compétences évaluées par Dr. Scratch. Ce sont des compétences très techniques, qui ne recoupent que partiellement les définitions de la pensée informatique de Brennan et Resnick (2012) ou Grover et Pea (2013). Ces compétences semblent parfois plus liées à la bonne utilisation du logiciel qu'à la pensée informatique elle-même. Certaines ne paraissent pas toujours pertinentes. Par exemple, l'interactivité du programme est un critère d'évaluation pris en compte par Dr. Scratch bien que ce critère ne semble pas directement relié à la maîtrise de la pensée informatique selon la grande majorité des auteurs ayant tenté de définir la pensée informatique.

Ce type d'outils est le plus souvent utilisé dans le cadre d'une évaluation formative-itérative qui consiste à évaluer les élèves pendant le processus d'apprentissage afin de leur permettre de développer et d'améliorer leurs compétences en pensée informatique en amélioration les programmes qui sont conçus. Ils sont conçus pour fonctionner avec un environnement de programmation particulier (et sont donc spécifiques à un langage ou à un logiciel), ce qui constitue également une limite à leur utilisation. Ils peuvent toutefois être utilisés à tous les âges, puisqu'ils se basent uniquement sur les programmes réalisés par les élèves. Par exemple, Dr. Scratch peut être adapté à tous les élèves qui ont appris à programmer sur Scratch, quel que soit leur niveau ou leur âge.

Des tâches de résolution de problèmes

Il existe d'autres démarches intéressantes qui diffèrent de la simple analyse du code produit par un élève et qui consistent en différentes tâches que l'élève doit accomplir. La réussite dans l'accomplissement de ces tâches est ainsi censée traduire la maîtrise de certains aspects de la pensée informatique. Grover, Cooper et Pea (2014) ont utilisé des captures d'écran de Scratch accompagnées de questions pour évaluer les apprentissages (« pourquoi ce pro-

gramme ne fonctionne pas ? » « Quand ce programme est exécuté, quelle valeur nous est renvoyée ? », etc.). Brennan et Resnick (2012) ont demandé aux élèves de partir d'un programme existant pour l'améliorer ou rajouter de nouvelles fonctionnalités définies par l'expérimentateur. De leur côté, Atmazidou et Demetriadis (2016) ont mis en place des exercices de résolution de problème qui impliqueraient l'utilisation de compétences identifiées comme faisant partie de la pensée informatique. Il était par exemple demandé d'identifier les structures communes qui guidaient le comportement d'un robot dans deux tâches différentes (abstraction), de proposer une solution plus générale à un problème (généralisation) etc. La pensée informatique étant très souvent associée au processus de résolution de problèmes, ce type d'outils d'évaluation semble pertinent, notamment dans le cadre d'activités de programmation pour lesquelles il est fréquent de considérer que les connaissances sont construites et évaluées par la résolution de problèmes (Rogalski & Samurçay, 1990).

L'une des tentatives les plus abouties en ce sens reste probablement le *Computational Thinking Test* ou *CTt* (González, 2015), dont la pertinence des items a été mesurée par un jury d'experts. Le *CTt* se compose de 28 items qui visent à estimer la maîtrise de certaines notions d'informatique : les instructions basiques, les boucles, les conditions et les fonctions. Le test est centré sur des exercices nécessitant la réalisation d'un parcours pour lesquels les participants ont un choix de réponse limité et prédéterminé. Ces réponses sont présentées sous forme de texte, de flèches ou de blocs similaires à ceux utilisés dans les logiciels de programmation visuels. Dans ces exercices, il est alternativement demandé au participant de trouver la séquence qui permet de résoudre le problème, de compléter l'instruction manquante dans un programme qui est proposé ou encore d'y trouver l'erreur qui empêche ce programme de répondre au problème posé.

Le test semble un excellent moyen de mesurer la compréhension de certaines notions fondamentales en programmation (« *computational concepts* », Brennan & Resnick, 2012) telles que les boucles ou les structures conditionnelles et la compréhension de leur fonctionnement. Son utilisation est généralement préconisée pour établir un diagnostic du niveau de l'élève à un moment donné et permet de comparer facilement les performances obtenues avant et après une intervention éducative (comparaison pré-test/post-test). Il présente par ailleurs une validité convergente, puisque les performances au *CTt* sont corrélées aux performances à d'autres tests mesurant les capacités spatiales, de raisonnement ou de résolution de problèmes (Roman-González *et al.*, 2017), ainsi que d'une validité prédictive vis-à-vis des performances académiques et de la réussite en programmation au collège (Roman-González, Pérez-González, Moreno-León & Robles, 2018). Enfin, il a été testé (et semble adapté) auprès des élèves dont le niveau se situe de la fin de l'école élémentaire jusqu'au début du lycée (de 9 à 16 ans).

Cependant, ce qui relève des notions et des savoirs semble confondu avec ce qui relève plus de la compétence et du savoir-faire dans ce CTt. González (2015) reconnaît que la structure du test, qui n'est composé que de questions

fermées à choix multiples, ne permet de mesurer le degré de maîtrise de la pensée informatique qu'en ce qui concerne ses aspects les plus simples, c'est-à-dire la capacité à comprendre les concepts mis en jeu et à reconnaître la solution à un problème posé parmi plusieurs possibilités données. Un instrument qui aurait pour objectif de mesurer également les aspects les plus complexes de la pensée informatique devrait alors demander à l'élève de montrer qu'il est capable d'assimiler et d'appliquer ces concepts pour résoudre des problèmes en proposant lui-même les solutions adéquates.

Certains outils proposent par ailleurs une évaluation de la pensée informatique fondée sur la résolution de problèmes indépendants des activités de programmation. Ces outils visent à évaluer dans quelle mesure les élèves sont capables de transférer leurs compétences en pensée informatique à une grande variété de problèmes, situations ou contextes nouveaux. C'est le cas notamment des épreuves du *Bebras International Contest* en Lituanie. La capacité à transférer ces compétences dans d'autres contextes, relatifs à des problèmes de la vie quotidienne, peut y être évaluée, y compris pour des élèves n'ayant jamais été initiés à la programmation. Roman-González *et al.* (2019) ont montré que le *CTt*, Dr. Scratch et les tâches Bebras peuvent être considérés comme des outils complémentaires, présentant des liens de corrélations et permettant d'évaluer différents niveaux d'acquisition de la pensée informatique.

Très récemment, plusieurs études ont proposé et validé de nouveaux outils visant à évaluer la capacité des élèves à résoudre des problèmes de programmation, en utilisant des langages « neutres », permettant à tous les élèves de réaliser ces tests, quels que soient les logiciels ou langages qu'ils ont appris à utiliser (Basu, Rutstein, Xu & Shear, 2020 ; Kong & Wang, 2021 ; Relkin, de Ruiter & Bers, 2020). Ces nouveaux outils ont été validés auprès d'élèves relativement jeunes, âgés de 5 à 11 ans. La résolution des problèmes présentés est ainsi censée révéler, non pas la maîtrise de certains concepts en informatique, comme pour le *CTt*, mais plutôt la maîtrise de certains processus cognitifs (abstraction, pensée algorithmique…) proches de ceux définis par Brennan et Resnick (2012) ou Selby et Woollard (2013) comme faisant partie de la pensée informatique.

À ce titre, le *Computational Thinking Assessment for Chinese Elementary Students* (*CTA-CES*, pas encore adapté dans les pays occidentaux) semble particulièrement prometteur (Li, Xu & Liu, 2021). Là encore, le cadre défini par Selby et Woolard (2013) pour décrire la pensée informatique constituait la base sur laquelle se sont appuyés les auteurs. Ceux-ci ont proposé un test comportant 25 items, similaires aux tâches Bebras dans le sens où les problèmes à résoudre ne sont pas des problèmes de programmation mais plutôt des problèmes de la vie quotidienne, à destination d'élèves âgés de 8 à 12 ans. En plus des méthodes psychométriques classiques utilisées pour valider cet outil, les auteurs ont réalisé des images par résonance magnétique (IRMf). Celles-ci ont montré que les activités neuronales des participants durant la réalisation de ce test et durant la réalisation d'activités de programmation sont corrélées. Ceci suggère que programmer et effectuer le CTA-CES sont deux activités qui impliquent les

mêmes processus cérébraux, ce qui constitue un indice supplémentaire en faveur de la validité de ce test, selon les auteurs.

Cependant, comme les autres tests évoqués précédemment, le CTA-CES ne propose que des questions à choix multiples et n'évalue donc pas la capacité des élèves à générer eux-mêmes leur propre solution à un problème. À notre connaissance, seuls Chen, Shen, Barth-Cohen, Jiang, Huang et Eltoukhy (2017) ont élaboré (et validé) un outil d'évaluation de la pensée informatique (selon la définition opérationnelle proposée par CSTA & ISTE, 2011) centré sur la résolution de problèmes et partiellement composé de questions ouvertes pour lesquelles les élèves (âgés de 9-10 ans) ne pouvaient pas choisir entre plusieurs solutions préétablies. Cependant, cet outil s'intègre dans le cadre d'un apprentissage de la programmation réalisé à l'aide de robots pédagogiques et semble difficile à adapter dans un autre contexte.

Les tâches de résolution de problèmes font généralement appel à des compétences qui dépassent le strict apprentissage de la programmation informatique. Pour cette raison, elles peuvent être utilisées auprès d'élèves n'ayant pas (encore) appris à programmer et se prêtent donc particulièrement bien aux comparaisons pré-test/post-test (évaluations avant et après une séquence pédagogique) qui permettent de facilement mesurer les bénéfices d'une intervention éducative.

Évaluer l'apprentissage « implicite » de la pensée informatique

Rowe *et al.* (2021) ont récemment proposé une approche innovante qui consiste à évaluer l'apprentissage implicite des élèves confrontés à une tâche faisant appel à certaines compétences liées à la pensée informatique. L'apprentissage implicite fait ici référence à un apprentissage dont la manifestation peut être faite lors de l'activité d'apprentissage elle-même (et non de manière différée avec des tests). L'outil présenté par Rowe *et al.* (2021) peut être considéré comme un outil de *data-mining* (exploration des données, en français) qui récupère et enregistre l'activité de l'élève en temps réel.

Les auteurs utilisent le jeu vidéo éducatif *Zoombinis*, composé d'épreuves basées sur la logique et la déduction. L'activité du joueur/élève pendant la résolution des problèmes posés est enregistrée sous la forme de données numériques qui peuvent ainsi être analysées et répliquées. Ici, ce n'est pas la capacité à résoudre les problèmes qui est évaluée mais bien les processus qui ont mené à cette résolution. À travers les données recueillies, les auteurs ont construit manuellement des outils permettant de détecter la maîtrise implicite de la pensée informatique des élèves âgés de 8 à 14 ans.

Les actions réalisées par l'élève lors de la résolution du problème permettent ainsi de mettre en évidence sa capacité à décomposer le problème en sous problèmes plus simples, à identifier une règle sous-jacente au problème, à tester de manière systématique différentes hypothèses ou encore à réutiliser certaines stratégies qui avaient fonctionné précédemment. Ces comportements (abstraction, décomposition…) sont caractéristiques de la pensée informatique selon

de nombreux auteurs (Brennan & Resnick, 2012 ; Selby & Woollard, 2013 ; Shute et al., 2017…).

Cette approche présente l'originalité de proposer une évaluation formative en temps réel fondée sur des activités ludiques, de mettre en évidence des comportements révélateurs d'une maîtrise émergente de la pensée informatique et de se placer aux antipodes des évaluations plus classiques sous forme de tests en ne s'appuyant sur aucune représentation visuelle ou verbale de la pensée informatique. Cependant, les connaissances techniques et le temps nécessaires pour récupérer et analyser les données du jeu et en déduire certaines compétences sont un véritable frein à la généralisation de cette méthode d'évaluation.

La nécessité d'évaluer distinctement la maîtrise de concepts et la compétence à résoudre des problèmes

L'identification des concepts et des compétences à résoudre des problèmes d'une certaine manière, qui constituent la maîtrise de la pensée informatique, est une première étape vers notre capacité à les évaluer. L'apprentissage peut être considéré comme un changement relativement permanent de nos connaissances (Mayer, 2011). Or, certains de ces changements ne sont pas directement observables : ils sont invisibles. Pour résoudre ce problème, Tricot et Musial (2020) proposent de considérer que l'apprentissage concerne un couple {tâche ; connaissance} dans lequel la connaissance est ce qui permet de réaliser une tâche et, réciproquement, la réalisation d'une tâche permet l'acquisition d'une connaissance. La réalisation d'une tâche au moment de l'évaluation peut donc être révélatrice d'une connaissance intrinsèquement invisible.

Les mêmes auteurs définissent la compétence comme un ensemble {Tâche ; Connaissances théoriquement nécessaires à la tâche ; Connaissances issues de la tâche}. La compétence correspond ainsi à la « capacité d'un individu à réaliser une certaine tâche et pour laquelle on peut décrire l'ensemble des connaissances nécessaires à la réalisation de cette tâche », certaines de ces connaissances étant spécifiquement issues de la pratique de la tâche. En suivant ce modèle, et en considérant que la pensée informatique est la compétence majeure que l'enseignement de la programmation cherche à développer, nous pouvons en tirer la conclusion suivante : la pensée informatique peut être considérée comme un ensemble regroupant une tâche (la résolution de problèmes algorithmiques), des concepts 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 (décomposition, généralisation, abstraction…).

Dans ce modèle, les connaissances pratiques issues des activités de programmation permettent le développement des compétences identifiées précédemment (notamment Selby & Woolard, 2013 ; Tchounikine, 2017) comme relevant de la maîtrise de la pensée informatique et de son utilisation. Ce modèle est également conforme à l'idée que la pensée informatique s'appuie

sur des concepts issus de l'informatique (Wing, 2006) en considérant que les notions fondamentales en programmation constituent les connaissances théoriques nécessaires à la résolution de problèmes algorithmiques, du moins dans le cadre de l'apprentissage de la programmation.

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'un côté les connaissances des élèves et d'un autre leur capacité à appliquer ces connaissances pour résoudre des problèmes complexes. Généralement, ces deux aspects fondamentaux sont confondus, si bien qu'on ne sait pas toujours si les outils d'évaluation proposés permettent de mesurer le niveau de compréhension des élèves vis-à-vis de certaines notions ou leur niveau de maîtrise d'une compétence qui leur permet de faire face à des situations inédites pour proposer des solutions en s'appuyant sur leurs connaissances. Or, nous considérons que cette distinction est capitale.

Si l'objectif est de développer des outils d'évaluation permettant de discriminer différents niveaux de maîtrise de la pensée informatique, alors il faut pouvoir en distinguer les aspects les plus fondamentaux, la connaissance et la compréhension des notions et concepts, 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. Une évaluation distincte de ces deux aspects de l'apprentissage de la programmation permettrait à l'enseignant d'affiner son diagnostic quant aux éventuelles difficultés rencontrées par ses élèves. Ceux-ci maîtrisent-ils suffisamment les concepts fondamentaux utilisés en programmation ? Si c'est le cas, sont-ils capables de les appliquer dans un contexte pratique en utilisant les stratégies adéquates ?

Cette distinction entre la maîtrise de concepts et compétence à résoudre des problèmes s'accorde d'ailleurs particulièrement bien avec les attendus de fin de cycle 4 (en France) en relation avec l'informatique et la programmation, bien que ceci ne constitue pas un argument scientifique en faveur de notre modèle. La capacité à « écrire, mettre au point et exécuter un programme » y figure ainsi que la maîtrise des « notions d'algorithme et de programme, notion de variable informatique, déclenchement d'une action par un événement, séquences d'instructions, boucles, instructions conditionnelles ».

Conclusion

L'apprentissage de la programmation est une discipline qui intègre ou réintègre progressivement les programmes scolaires pour répondre aux besoins éducatifs entraînés par les changements de notre société. La programmation semble potentiellement indiquée pour acquérir les concepts de la « pensée informatique », une façon de concevoir des systèmes et de résoudre des problèmes, proche de l'informatique et de la méthode scientifique, dont le développement

et la maîtrise constitueraient un enjeu sociétal majeur du XXIe siècle. Cependant, la définition même de la pensée informatique ne fait pas l'objet d'un consensus entre les chercheurs. D'où le problème de l'évaluation de sa maîtrise qui en découle.

Diverses possibilités sont proposées aux enseignants pour tenter de mesurer les compétences en pensée informatique de leurs élèves. Parmi celles-ci, on retrouve des échelles, telles que celle intitulée *Computational Thinking Scales* qui propose d'évaluer certaines facultés identifiées comme proche de la pensée informatique (capacité de résolution de problèmes, pensée créative…) et qui s'adressent plutôt aux élèves les plus âgés (fin du lycée). Le caractère auto-évaluatif de ces échelles demeure cependant une limite importante à cette approche.

Certains outils, comme Dr. Scratch, peuvent également analyser « objectivement » et automatiquement des projets réalisés sur le logiciel Scratch pour estimer la maîtrise de la pensée informatique. Néanmoins, cette approche se limite finalement à l'observation des artefacts d'apprentissage des élèves et ne peut suffire à acquérir une vision globale de leur niveau.

Une autre possibilité consiste à soumettre les élèves à des exercices de résolution de problèmes. À ce titre, le *Computational Thinking Test* semble l'outil le plus fiable aujourd'hui pour permettre d'évaluer la maîtrise de certaines notions fondamentales en programmation. De nombreux outils d'évaluation, basés sur des exercices de résolution de problèmes, sont également disponibles pour mettre en évidence la maîtrise de certains processus cognitifs décrits comme des composantes de la pensée informatique. Cependant, la plupart d'entre eux ne comporte que des questions à choix multiples et ne permettent donc pas d'évaluer la capacité des élèves à générer eux-mêmes leur propre solution à un problème donné.

Enfin, une dernière solution consiste à évaluer non pas la capacité à réaliser une tâche mais les plutôt les processus qui ont mené à la réalisation de cette tâche et qui, dans le cadre de jeux basés sur la résolution de problèmes logiques, peuvent être révélateurs d'une certaine maîtrise de la pensée informatique. Mais cette solution paraît difficile à généraliser à grande échelle.

Ces outils peuvent bien sûr être combinés pour garantir une évaluation plus précise des compétences. Cependant, il subsiste encore de larges incertitudes sur la manière d'évaluer l'acquisition de la pensée informatique auprès d'enfants et d'adolescents, dans le cadre d'activités de programmation. De nouveaux outils, qui distinguent d'une part la compréhension des notions fondamentales en programmation et d'autre part la capacité à résoudre des problèmes algorithmiques, doivent être proposés.

Références

Aho, A. V. (2012). Computation and computational thinking. *The computer journal*, *55*(7), 832-835.

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), 17-39.

Basu, S., Rutstein, D., Xu, Y., & Shear, L. (2020, February). A principled approach to designing a computational thinking practices assessment for early grades. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 912-918).

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education-Implications for policy and practice* (No. JRC104188). Joint Research Centre (Seville site).

Bourgeois, A., Birch, P., & Davydovskaia, O. (2019). *Digital Education at School in Europe. Eurydice Report*. Education, Audiovisual and Culture Executive Agency, European Commission.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65-72).

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* (pp. 1-25). Vancouver, Canada.

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162-175.

CSTA & ISTE (2011). Operational Definition of Computational Thinking for K–12 Education. *National Science Foundation*.

del Olmo-Muñoz, J., Cózar-Gutiérrez, R., & González-Calero, J. A. (2020). Computational thinking through unplugged activities in early years of Primary Education. *Computers & Education*, *150*, 103832.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43.

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).

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*(3), 583.

Kong, S. C., & Wang, Y. Q. (2021). Item response analysis of computational thinking practices: Test characteristics and students' learning abilities in visual programming contexts. *Computers in Human Behavior*, *122*, 106836.

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.

Korkmaz, Ö., & Xuemei, B. A. İ. (2019). Adapting computational thinking scale (CTS) for Chinese high school students and their thinking scale skills level. *Participatory Educational Research*, *6*(1), 10-26.

Kukul, V., & Karatas, S. (2019). Computational thinking self-efficacy scale: Development, validity and reliability. *Informatics in Education*, *18*(1), 151-164.

Li, Y., Xu, S., & Liu, J. (2021). Development and validation of computational thinking assessment of Chinese elementary school students. *Journal of Pacific Rim Psychology*, *15*, 18344909211010240.

Lodi, M., & Martini, S. (2021). Computational thinking, between Papert and Wing. *Science & Education*, *30*(4), 883-908.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, *41*, 51-61.

Mayer, R. E. (2011). Does styles research have useful implications for educational practice? *Learning and Individual Differences*, *21*(3), 319-320.

Ministère de l'Education nationale, de l'enseignement supérieur et de la recherche (2015). Socle commun de connaissances, de compétences et de culture. Décret nº 2015-372. *Bulletin officiel de l'Éducation nationale, de l'enseignement supérieur et de la recherche, 23 avril*.

Mohaghegh, D. M., & McCauley, M. (2016). Computational thinking: The skill set of the 21st century. *International Journal of Computer Science and Information Technologies*, *7*(3), 1524-1530.

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).

Moreno-León, J., Román-González, M., Harteveld, C., & Robles, G. (2017, May). On the automatic assessment of computational thinking skills: A comparison with human experts. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 2788-2795).

Nogry, S. (2018). Comment apprennent les élèves au cours d'une séquence de robotique éducative en classe de CP ?. *De 0 à 1 ou l'heure de l'informatique à l'école* (pp. 235-246). Lausanne : Peter Lang.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365-376.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*(5), 583-602.

Relkin, E., de Ruiter, L., & Bers, M. U. (2020). TechCheck: Development and validation of an unplugged assessment of computational thinking in early childhood education. *Journal of Science Education and Technology*, *29*, 482-498.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, *13*(2), 137-172.

Rogalski, J., & Samurçay, R. (1990). Acquisition of programming knowledge and skills. In *Psychology of programming* (pp. 157-174). Academic Press.

Rogalski, J., Samurçay, R., & Hoc, J. M. (1988). L'apprentissage des méthodes de programmation comme méthodes de résolution de problème. *Le travail humain*, *51*(4), 309-320.

Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction*, *18*, 47-58.

Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining assessment tools for a comprehensive evaluation of computational thinking interventions. In *Computational thinking education* (pp. 79-98). Springer, Singapore.

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. In *Educode-Conférence internationale sur l'enseignement au numérique et par le numérique*.

Rowe, E., Almeda, M. V., Asbell-Clarke, J., Scruggs, R., Baker, R., Bardar, E., & Gasca, S. (2021). Assessing implicit computational thinking in Zoombinis puzzle gameplay. *Computers in Human Behavior*, *120*, 106707.

Samurçay, R., & Rouchier, A. (1985). De « faire » à « faire faire » : planification d'actions dans la situation de programmation. *Enfance*, *38*(2), 241-254.

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2019). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, *111*(5), 764.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition. In *Paper presented at the 18th annual conference on innovation and technology in computer science education, Canterbury*.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

Sigayret, K., Tricot, A., & Blanc, N. (2022). Unplugged or plugged-in programming learning: A comparative experimental study. *Computers & Education*, 104505.

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798.

Tchounikine, P. (2017). Initier les élèves à la pensée informatique et à la programmation avec Scratch. *Laboratoire d'informatique de Grenoble*. En ligne : http://ligmembres.imag.fr/tchounikine/PenseeInformatiqueEcole.html.

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

Van Lint, S. (2016). La notion de compétence et son évaluation. *MARS*. Technologie, *Sciences et techniques industrielles*, 202, 30-33.

Weese, J. L., & Feldhausen, R. (2017, June). STEM outreach: Assessing computational thinking and problem solving. In *2017 ASEE Annual Conference & Exposition*.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of science education and technology*, *25*(1), 127-147.

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

Wing, J. M. (2011). Research Notebook: Computational Thinking. What and Why? *The Link*. Pittsburg : Carnegie Mellon.

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607.