Formation et profession
Revue scientifique internationale en éducation

Pensée informatique : portrait conceptuel des aspects inhérents à la programmation en contexte scolaire
Computational Thinking: Conceptual Overview of Components Inherent to Programming in School Settings

Simon Parent

Volume 30, Number 2, 2022

Des usages numériques multiples et variés en contexte québécois

URI: https://id.erudit.org/iderudit/1098055ar
DOI: https://doi.org/10.18162/fp.2022.651

See table of contents

Publisher(s)

Centre de recherche interuniversitaire sur la formation et la profession enseignante (CRIFPE)

ISSN

1718-8237 (print)
2368-9226 (digital)

Explore this journal

Cite this article

Parent, S. (2022). Pensée informatique : portrait conceptuel des aspects inhérents à la programmation en contexte scolaire. *Formation et profession*, *30*(2), 1–15. https://doi.org/10.18162/fp.2022.651

Article abstract

This article offers a brief conceptual portrait of computational thinking, a concept that is highly recurrent in the literature and that is subject to numerous interpretations. A non-exhaustive review of the literature has therefore allowed us to clarify what computational thinking is, while specifying the role of many other related concepts evoked in the works of the authors studied: algorithmic thinking, abstraction, etc. The presentation of integrative models offers an interesting and synthetic perspective on this phenomenon that is becoming more and more widespread in the educational field.

Pensée informatique : portrait conceptuel des aspects inhérents à la programmation en contexte scolaire

Simon **Parent**
Université de Montréal (Canada)

Computational Thinking: Conceptual Overview of Components Inherent to Programming in School Settings

doi: 10.18162/fp.2022.651

Résumé

Cet article offre un bref portrait conceptuel de la pensée informatique, un concept hautement récurrent dans la littérature et qui renvoie à de nombreuses interprétations. Une recension non exhaustive de la littérature nous a donc permis d'apporter plusieurs clarifications sur ce qu'est la pensée informatique, tout en précisant le rôle de nombreux autres concepts voisins évoqués dans les travaux des auteurs étudiés : pensée algorithmique, abstraction, etc. La présentation de modèles intégrateurs offre une perspective intéressante et synthétique sur ce phénomène de plus en plus répandu dans le milieu de l'éducation.

Mots-clés

Programmation, pensée informatique, pensée algorithmique, résolution de problèmes, algorithmes.

Abstract

This article offers a brief conceptual portrait of computational thinking, a concept that is highly recurrent in the literature and that is subject to numerous interpretations. A non-exhaustive review of the literature has therefore allowed us to clarify what computational thinking is, while specifying the role of many other related concepts evoked in the works of the authors studied: algorithmic thinking, abstraction, etc. The presentation of integrative models offers an interesting and synthetic perspective on this phenomenon that is becoming more and more widespread in the educational field.

Keywords

Programming, computational thinking, algorithmic thinking, problem solving, abstraction, algorithms.

Introduction

À l'instar de nombreux pays et systèmes éducatifs internationaux, le Québec s'est doté en 2018 d'un Plan d'action numérique en éducation et en enseignement supérieur (PANEES) (Ministère de l'Éducation et de l'Enseignement supérieur, 2018). Ce dernier présente 33 mesures articulées par trois grandes orientations : développement de la compétence numérique, le numérique comme vecteur d'apprentissage et création d'un environnement propice au déploiement du numérique. C'est d'ailleurs dans ce document que le Gouvernement du Québec évoque l'intérêt d'« accroître l'usage pédagogique de la programmation informatique » (2018, p. 27), et ce, pour amener l'élève à réaliser des apprentissages et à développer des compétences.

Dans une perspective pédagogique, la programmation peut être utilisée dans le but de développer une compréhension générale du processus de programmation en soi (quel que soit le langage) ou encore de représenter un vecteur de développement de compétences diverses. Papert (1981) parlait d'apprendre les rudiments de la programmation tout en aspirant à des impacts positifs sur la façon d'apprendre des élèves dans toutes les matières.

Parmi ces dernières, nous retrouvons la pensée informatique, le concept qui est au cœur de cet article et que nous déclinerons dans les prochaines sections. Ces compétences peuvent à leur tour être réinvesties tant dans les autres matières scolaires que pour la résolution de problèmes complexes dans la vie de tous les jours, par exemple. Au cours des dernières années sont apparues de nombreuses applications en ligne et gratuites permettant aux élèves du primaire et du secondaire de réaliser et mettre en œuvre des programmes informatiques relativement complexes et variés. Il est alors difficile

ISSN 2368-9226

de ne pas considérer les potentialités de la programmation en tant qu'outil d'apprentissage, où de petits personnages attachants et des mises en situation adaptées aux intérêts des jeunes contribuent au ludisme de l'activité. La programmation devient ainsi un vecteur pour le développement d'une variété d'habiletés et de savoir-faire qui peuvent être mobilisés dans plusieurs autres contextes et disciplines scolaires.

Devant le nouvel intérêt politique pour l'intégration de la programmation dans le cursus scolaire québécois, nous avons parcouru la littérature sur la programmation en contexte scolaire. C'est ainsi que nous avons rapidement constaté un paysage conceptuel pluriel, tout en notant la présence d'un concept fondamental : la pensée informatique. Considérant cet amas conceptuel aux multiples facettes, nous avons senti le besoin de mettre en contraste la programmation et la pensée informatique, en précisant : 1) ce qu'est la programmation ; 2) ce qu'est la pensée informatique et les concepts qui y sont étroitement associés, notamment dans le champ des mathématiques ; 3) quels modèles offrent une vision intégratrice de la pensée informatique et des concepts associés. Ces trois volets structurent cet article.

Méthode

Nous présentons ci-après les éléments conceptuels et théoriques tirés d'une revue de la littérature. Bien que ce processus n'ait pas fait l'objet d'une systématisation, il a néanmoins permis de consulter les écrits phares dans la littérature scientifique à l'égard de la pensée informatique et de la programmation (en contexte scolaire).

Nous avons utilisé les bases de données et les moteurs de recherche suivants : *Google Scholar*, *Scopus*, *Publish or Perish* ainsi que *EduTechLib*. Les mots-clés utilisés dans le cadre de nos recherches étaient, en français : programmation ; pensée informatique ; pensée computationnelle ; école ; primaire ; secondaire, et en anglais : coding; computationnal thinking; school; K-12. Nous avons également consulté des ouvrages disponibles aux bibliothèques facultaires de l'Université de Montréal. Nous n'avons pas utilisé de critère d'inclusion ou d'exclusion défini, mais avons favorisé des écrits relativement récents, c'est-à-dire à compter de 2000. Cela dit, des travaux moins récents ont été inclus en raison de leur importance dans la littérature (par ex. : Papert, 1981). Les ouvrages trouvés et jugés pertinents (N = 44), après une analyse sommaire du titre et du résumé, ont été consignés dans un dossier correspondant à la catégorie appropriée sous la forme d'une notice bibliographique. Nous avons ensuite parcouru chaque source afin d'en faire ressortir les principaux éléments d'information. Les sources ont alors été intégrées dans un tableau de consignation à l'aide du logiciel Excel.

La programmation : outil de médiation humain-ordinateur

Le fonctionnement de l'ordinateur se base essentiellement sur le traitement et la mémorisation d'informations (Bossuet, 1982 ; Guttag, 2017). N'ayant pas la capacité de réfléchir, l'ordinateur doit être contrôlé par des indications claires, précises et ne laissant pas de place à l'interprétation (Hardouin-Mercier, 1974). Cette affirmation pourrait être discutable au regard des récentes avancées de l'intelligence artificielle, mais nous évitons sciemment d'aborder cette sphère de l'informatique qui ne pourrait être brièvement résumée, au risque d'en tracer un portrait inexact. Cela dit, le moyen employé pour transmettre des indications claires et objectives à un ordinateur est la programmation.

La programmation est un procédé permettant de transmettre des consignes claires à un agent de traitement de l'information dans un but précis (Turski, 1978). Le programme constitue donc l'ensemble des consignes qui sont soumises à l'ordinateur en vue de lui faire exécuter une tâche en particulier. L'activité de programmation se décline en deux volets : l'acte de programmer et l'acte de coder. Alors que l'acte de programmer consiste à élaborer une structure complexe et organisée de consignes, c'est-à-dire un algorithme (Henri, 2014 ; Rice et Desautels, 1969) permettant de mettre en œuvre les différentes fonctionnalités de la machine (Hardouin-Mercier, 1974 ; Van Roy et Haridi, 2004), l'acte de coder fait plutôt référence à la traduction de ces séries de consignes en un « langage parfaitement formalisé » qui pourra être compris par l'ordinateur (Baron et Bruillard, 2001 ; Lopez, 1986). Bien que la programmation soit souvent associée au simple fait d'écrire des lignes de codes, on nous rappelle qu'il est important de la considérer comme un processus complet (Mannila et al., 2014), un « exercice beaucoup plus large qui englobe des activités de conception, d'écriture, de test et de maintenance ; elle s'apparente autant, voire davantage, à la résolution de problèmes qu'à l'écriture d'un code » (Henri, 2014, p. 11). Un point de vue aussi partagé par Mannila et ses collègues :

> If programming is seen as merely "coding", a routine job of giving instructions to the computer in order to solve a set of traditional problems, we do not believe it is enough. Coding is a fraction of the programming process, and can be seen as a task that you do last in order to implement the solution that you have ended up with through other phases such as analysis, decomposition and design, (Mannila et al., 2014, p. 4)

Cette idée de complexité inhérente à la programmation et à la résolution de problèmes avait d'ailleurs été soulevée par Seymour Papert, l'un des précurseurs de la programmation pédagogique : « quand on apprend à programmer un ordinateur, on n'y arrive presque jamais du premier coup. Apprendre à passer maître en l'art de programmer, c'est devenir hautement habile à déceler où se nichent les "bugs" et à y remédier » (Papert, 1981). Nous retenons donc que la programmation est une activité en deux phases (programmer et coder), qu'elle sert de médiation entre l'humain et l'ordinateur, et qu'elle est associée de près à la résolution de problèmes. Ce processus complexe et structuré peut être représenté à l'aide d'outils comme l'ordinogramme (Hardouin-Mercier, 1974 ; Henri, 2014), ou organigramme, qui met en évidence les liens logiques et conditionnels entre les différentes parties d'un programme. Ces ordinogrammes peuvent ensuite être traduits de façon textuelle, par le codage, en un langage spécifique de programmation. Il existe d'ailleurs plusieurs langages de programmation (par ex. : Python, C, C++, etc.) qui ont tous la même utilité, mais qui ne sont pas construits de la même façon, au même titre que les langues parlées à travers le monde. Un même algorithme pourra donc être écrit à l'aide de différents codes, menant toutefois à un même résultat. Avec ces nombreux niveaux de complexité, la programmation devient un outil singulièrement intéressant pour le développement de la pensée informatique (Grover et Pea, 2013).

La pensée informatique : un paysage conceptuel pluriel

Le fait de vouloir apporter des nuances entre la programmation et la pensée informatique génère son lot de conséquences. L'une d'elles est la nécessité de bien définir ces deux concepts. Or, bien qu'il semble exister une définition globale et plutôt consensuelle de la programmation, il en va autrement pour la pensée informatique. En effet, ce terme renvoie à un champ conceptuel très vaste et pluriel,

où l'absence de consensus, l'utilisation de termes synonymes et les divergences entre la littérature francophone et anglophone en complexifient la compréhension.

Du côté francophone, nous identifions trois termes dont les frontières sont plus ou moins définies et qui semblent, à première vue, désigner des objets très similaires : pensée informatique, pensée algorithmique et pensée computationnelle. Du côté anglophone, les deux termes existants sont *computational thinking* et *algorithmic thinking*. Nous avons utilisé le moteur de recherche *Google Scholar* pour offrir un aperçu quantitatif et approximatif de la littérature scientifique (tableau 1).

Tableau 1
Fréquence des résultats sur Google Scholar[1]

Terme	Fréquence
Computational thinking	2 100 000 résultats
Algorithmic thinking	172 000 résultats
Pensée informatique	116 000 résultats
Pensée algorithmique	20 500 résultats
Pensée computationnelle	6 070 résultats

Les résultats de cette analyse sommaire démontrent que la littérature anglophone est considérablement plus fournie dans ce champ, en termes quantitatifs. De plus, la faible fréquence de résultats associés au terme « pensée computationnelle » nous amène à penser qu'il s'agit d'un calque de l'anglais et qu'il est entièrement assimilable au concept de pensée informatique. D'ailleurs, la traduction littérale à l'anglais de « pensée informatique » est computational thinking, ce qui a été explicitement accepté par Romero (2017) et Tchounikine (2016). Les références à la pensée computationnelle, dans la littérature francophone, ont donc été incluses dans notre explicitation du concept de pensée informatique. Afin de préciser l'ambiguïté présente dans la littérature entre les pensées informatique (computational) et algorithmique (algorithmic), nous les déclinons ci-dessous.

La pensée informatique (computational thinking)

Dans un article paru en 2006, intitulé *Computational Thinking*, Jeannette Wing, professeure d'informatique à l'Université Columbia, parle de la pensée informatique (PI) en ces mots : « it represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use » (Wing, 2006, p. 33). Elle poursuit ensuite en suggérant une longue définition de la pensée informatique qui deviendra, au fil des années, l'une des principales sources d'information en la matière. L'auteure énonce une définition plus synthétique dans un article ultérieur : « Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent » (Wing, 2010). De cette définition, nous tirons trois éléments fondamentaux de la PI. D'abord, il s'agit de plusieurs processus de pensée, c'est-à-dire d'un ensemble de démarches abstraites et complexes. Puis, nous notons l'interdépendance établie entre la PI et le processus de résolution de problème (RP). Enfin,

ce qui caractérise particulièrement la PI est sa finalité, c'est-à-dire l'opérationnalisation d'un processus abstrait afin qu'il puisse être interprété par un agent de traitement de l'information, autrement dit, un ordinateur. Cela étant dit, les concepts présentés par Wing n'étaient alors pas nouveaux. En effet, déjà au début des années 1970 en France dans le cadre du Séminaire de Sèvres, « ce qui était important [dans l'introduction de l'enseignement de l'informatique au secondaire] était, non pas l'ordinateur, mais bien la démarche informatique que l'on peut caractériser comme algorithmique, opérationnelle, organisationnelle » (« L'enseignement de l'informatique à l'école secondaire, » 1970, p. 33).

De nombreux auteurs ont par la suite tenté d'approfondir le champ de la PI. Les travaux de Selby et Woollard (2013) offrent un regard global intéressant à ce propos. En effet, une revue de la littérature leur a permis d'identifier des composantes de la PI telles qu'identifiées dans les 22 articles qu'ils ont retenus. Ils ont ainsi pu offrir une définition synthétique de la PI qui soit en cohérence avec la littérature scientifique :

> […] computational thinking is an activity, often product oriented, associated with, but not limited to, problem solving. It is a cognitive or thought process that reflects the ability to think in abstractions, to think in terms of decomposition, to think algorithmically, to think in terms of evaluations, and to think in generalizations. (Selby et Woollard, 2013, p. 5)

On y retrouve en essence les éléments fondamentaux évoqués par Wing, en apportant une certaine précision par l'énumération d'habiletés (i.e. penser en termes de…). La résolution de problèmes y occupe une place centrale, comme c'est le cas pour plusieurs auteurs (Aho, 2012 ; Barr et Stephenson, 2011 ; Curzon et McOwan, 2017 ; Delcker et Ifenthaler, 2017 ; Hu, 2011 ; Tchounikine, 2016 ; Voogt et al., 2015). La littérature semble donc positionner la PI à la rencontre de la RP et de l'informatique, ce qui n'est pas sans rappeler l'abstraction, qui est aussi une composante majeure de la PI pour plusieurs auteurs (Caspersen et Nowack, 2013 ; Hromkovič et al., 2016 ; Hu, 2011 ; Romero, 2017 ; Serafini, 2011 ; Voogt et al., 2015). La très grande complexité de la PI et son rôle central en contexte de RP suscitent l'abstraction :

> […] expressed in the context of problem solving, abstraction corresponds to the question "Can we adapt an already known or universally available strategy to solve the problem at hand?" Once we know how to solve a single instance, we then employ the concept of automation to apply our solution to a large set of instances. (Hromkovič et al., 2016, p. 113)

Dans la PI, l'abstraction est mobilisée avec la définition de régularités (Wing, 2010), les modèles et la simulation (ISTE et CSTA, 2011 ; Voogt et al., 2015), ainsi que la systématisation (Tatar et al., 2017). Par ailleurs, certains auteurs ont orienté leur réflexion de façon plus pragmatique en vue d'une application en contexte scolaire. Ainsi, dans une perspective évaluative, Romero (2017) a proposé 6 composantes opérationnalisables de la PI :

> (1) capacité à identifier des objets et des itérations (analyse/abstraction), (2) capacité à organiser et modéliser des données de manière efficace (organisation/modélisation), (3) comprendre la logique d'un algorithme (littératie numérique), (4) capacité à créer un programme informatique (programmation), (5) compréhension, analyse critique et vision technologique (vision critique), (6) capacité à développer des projets créatifs à travers la programmation (technocréativité). (Romero, 2017, p. 9)

Ces composantes sont similaires à celles qui ont été évoquées dans les autres définitions présentées, mais nous pouvons ici voir s'ajouter la programmation, qui n'avait pas été soulignée jusqu'à présent. De façon complémentaire, les travaux de Brennan et Resnick (2012), qui ont élaboré le *Computational Thinking Framework* (tableau 2), offrent une vision intégrée de la pensée informatique et de la programmation, tout en s'inscrivant dans une visée évaluative opérationnalisable.

Tableau 2

Cadre conceptuel de la pensée informatique selon Brennan et Resnick (2012)

Computational thinking concepts	Sequences, loops, parallelism, events, conditionals, operators, and data.
Computational thinking practices	Being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing.
Computational thinking perspectives	Expressing, connecting, questioning.

Brennan et Resnick proposent donc une vision pragmatique de la pensée informatique en y énonçant des concepts et des pratiques (practices) à des fins d'évaluation, en plus d'être fortement associés à des pratiques de programmation, bien qu'ils mentionnent leur application en contextes de programmation ou non (programming and non-programming contexts). Tchounikine (2016) s'inscrit aussi dans cette perspective en définissant la PI comme étant une façon « d'appréhender le monde selon l'approche employée en programmation par les développeurs de logiciels » (p.3), en insistant lui aussi sur la possibilité de mobiliser la PI sans utiliser un ordinateur, de façon déconnectée (unplugged).

Ces auteurs mettent donc en lumière le lien fort unissant la programmation à la pensée informatique. Cela dit, selon la plupart des auteurs, la PI n'est pas limitée à l'activité de programmation. En effet, la PI représenterait un ensemble d'habiletés qui peuvent être mobilisées dans plusieurs contextes (Delcker et Ifenthaler, 2017), ayant un effet transversal à travers les disciplines (Curzon et McOwan, 2017). Il est même suggéré que le simple fait de favoriser la systématisation pourrait contribuer au développement d'une PI prototypique (*proto-computational thinking*) chez des élèves du primaire (Tatar et al., 2017). Enfin, Romero (2018) tisse un lien fort entre l'apprentissage de la PI et le rapport qu'entretiennent les individus avec l'intelligence artificielle (IA). En effet, elle avance qu'une PI développée permettrait d'envisager l'IA non pas comme une « boîte noire pleine de mystères », mais plutôt de l'aborder de façon critique et réfléchie. Ces propos ne sont pas sans rappeler ceux de Mirabail, qui affirmait : « posséder une culture informatique, c'est pouvoir agir le moment venu en [personne avertie, compétente], responsable » (Mirabail, 1990, p. 14).

En somme, nous constatons que la PI est en fait un ensemble de processus de pensée complexes qui ne se limitent pas à la programmation ou même à l'informatique, mobilisant notamment la compétence de résolution de problèmes. Les principaux concepts clés qui y sont associés sont l'abstraction, la systématisation (généralisation), la programmation et l'algorithme. Qu'en est-il de la pensée algorithmique ?

La pensée algorithmique (algorithmic thinking)

Alors que certains auteurs subordonnent la pensée algorithmique (PA) à la PI (Curzon et McOwan, 2017 ; Wing, 2010 ; Tchounikine, 2016), plusieurs autres auteurs étudiant la PI ne mentionnent en aucun moment la PA. Est-ce alors que la PA est déclinée sans être nommée, ou alors qu'elle est exclue sciemment ? Une exploration de la littérature associée à la PA nous a amené à découvrir un champ fortement rattaché aux sciences et aux mathématiques (Modeste, 2012 ; Sengupta et al., 2013 ; Weintrop et al., 2016).

Avant même de chercher à comprendre ce qu'est la pensée algorithmique, nous croyons nécessaire de définir ce qu'est un algorithme dans cette littérature spécifique. Selon Rice et Desautels (1969), il s'agit d'une recette, souvent utilisée pour résoudre un problème, qui est monosémique, non ambiguë et qui doit finir par s'arrêter à un moment ou à un autre. Cette définition de l'algorithme n'est pas sans rappeler la définition de la programmation, plus particulièrement l'acte de coder à l'aide d'un langage informatique, et la définition de la PI selon Selby et Woollard (2013), où la résolution de problème est au cœur du processus. Alors, quelle est donc la particularité de la PA ? La réponse à cette question réside en partie dans le fait qu'elle pourrait être considérée comme l'expression du lien puissant qui unit la mathématique à l'informatique (Briant et Bronner, 2015 ; Hu, 2011). De nombreux auteurs soulignent la grande proximité entre la programmation et les mathématiques (Knuth, 1985 ; Lu et Fletcher, 2009 ; Parmentier, 2018). Ce rapprochement est soulevé clairement par Modeste : « la pensée algorithmique peut être vue comme faisant partie de la pensée mathématique […] cependant, voir la pensée algorithmique comme pensée majeure de l'informatique permet un réel enrichissement de son analyse » (Modeste, 2012, p. 477). Hromkovič et ses collègues (2016) affirment d'ailleurs que la PA est le cœur de la science informatique, fortement rattachée à la programmation (Futschek et Moschitz, 2011 ; Lamagna, 2015). Nous avons synthétisé et schématisé cette conjoncture complexe pour en simplifier la compréhension (figure 1).

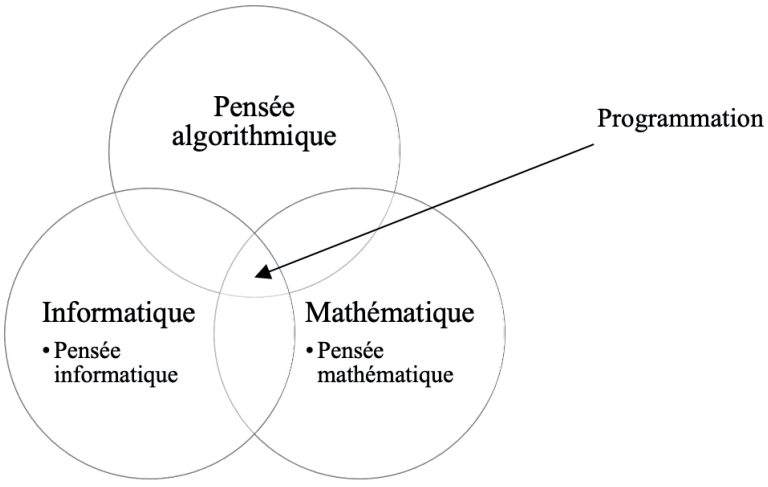

Figure 1
Rapprochements entre la PA, l'informatique et la mathématique dans la littérature

Nous avons créé ce diagramme de Venne afin de démontrer que ces trois entités existent de façon distincte, mais qu'elles se rejoignent. Il illustre aussi que la programmation appartient à deux champs qui se complémentent, nommément l'informatique et la mathématique. Bien entendu, cette simplification permettant une meilleure compréhension comporte aussi le risque d'hypersimplifier une réalité complexe. Nous considérons toutefois qu'il résume la littérature que nous avons consultée et qu'il est fidèle aux propos des auteurs que nous avons cités. Par ailleurs, pour ce qui est des composantes de la PA, Futschek et Moschitz suggèrent que cette dernière est en fait une compétence étroitement liée à la RP dont les habiletés sont les suivantes :

> […] analyze given problems, specify problems precisely, find the basic actions that are adequate to given problems, construct correct algorithms to given problems using the basic actions, think about all possible special and normal cases of a problem, evaluate algorithms (correctness, efficiency, termination) [and] improve the efficiency of algorithms. (Futschek et Moschitz, 2010, p. 2)

Lamagna (2015) définit quant à lui la PA comme étant composée d'habiletés liées aux procédures informatiques : la compréhension, l'exécution, la création et l'évaluation de l'efficacité de ces procédures, qui devraient être en mesure de résoudre systématiquement le problème posé. Dans les deux cas, la RP est centrale à la PA, comme pour la PI.

De la pluralité à la vision intégrée

Il peut sembler difficile de jongler avec les concepts de PI, de PA et de RP. Cela dit, plusieurs auteurs ont explicité les relations unissant ces concepts à l'aide de modèles intégrateurs. D'abord, les travaux de McMaster, Rague et Anderson (2010) tracent un cadre, le *MAC Thinking Framework* (figure 2), décliné en trois volets : le monde réel (*Real World*) dans lequel se situent le problème à résoudre et ses données, le monde abstrait (*Abstract World*) dans lequel se conçoit un algorithme permettant de résoudre le problème, puis le monde informatique (*Computer World*) dans lequel l'algorithme est appliqué, à l'aide d'un ordinateur, pour solutionner le problème, ce qui peut notamment être fait à l'aide de la programmation.

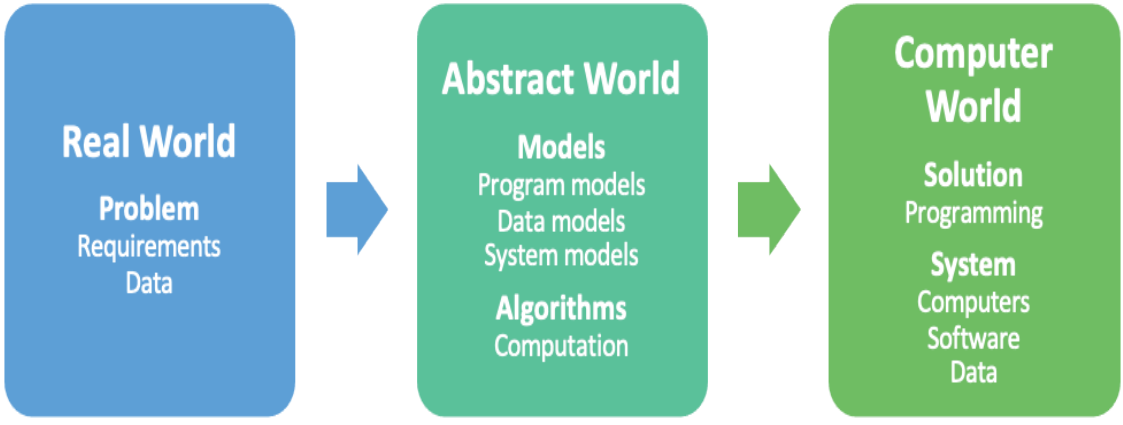

Figure 2
MAC Thinking Framework de McMaster et al. (2010)

Ils tracent ainsi un processus chronologique se terminant par l'utilisation potentielle de la programmation. Un processus similaire est présent dans les travaux de Briant et Bronner (2015), qui explorent les liens entre la mathématique, l'algorithmique et l'informatique, et ce, dans un contexte général de RP. Ils ont représenté à l'aide d'un schéma ce qu'ils appellent la « double transposition », c'est-à-dire le passage d'un problème mathématique à une application informatique de programmation (figure 3).

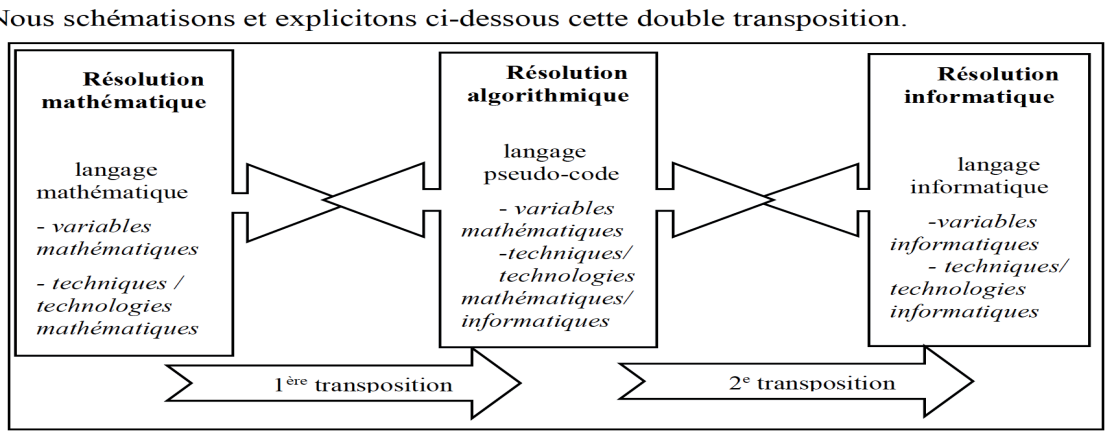

Figure 3
La double transposition, selon Briant et Bronner (2015)

Nous voyons ici aussi se dessiner un processus chronologique itératif dans lequel l'apprenant devra cheminer afin de résoudre un problème, tout comme le suggère le modèle de McMaster et al. (2010). Il existe aussi des modèles semblables adaptés à différents secteurs, comme celui de Sengupta et al. (2013), destiné à l'enseignement des sciences au primaire et au secondaire. Nous avons également retenu le modèle de Romero (2016), qui offre une perspective récente, concrète et appliquée de la PI en contexte authentique (figure 4), se rapprochant ainsi de notre intérêt pour la programmation en contexte scolaire. Bien que ce modèle ne fasse pas référence explicitement à la PA, nous voyons tout de même le lien unissant la PI à la RP représenté par la jonction entre le cercle bleu clair et le cercle vert.

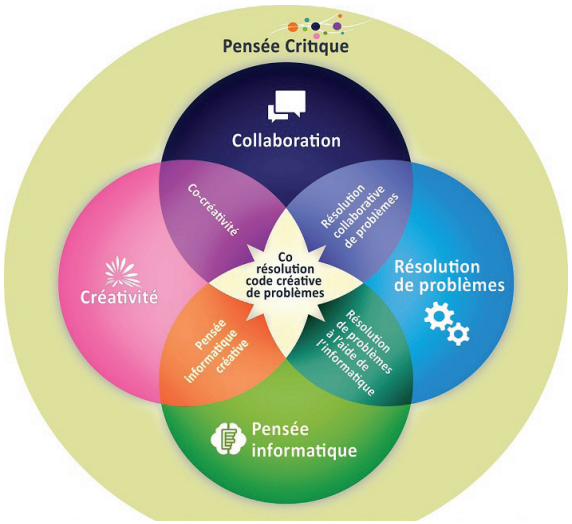

Figure 4
Modèle de co-résolution code-créative de situations problèmes de Romero (2016)

Romero a aussi choisi de juxtaposer la collaboration à la RP et à la PI, un point de vue entre autres partagé par Kazimoglu et al. (2012), qui ont défini la «l'aspect social» comme étant l'une des cinq compétences fondamentales associées à la PI.

Les auteurs évoqués démontrent donc la grande proximité entre la PI et la PA, de même que l'omniprésence de la RP. Or, lorsqu'il est question de PA, les auteurs évoquent des composantes qui font toutes partie de la PI, ce qui n'est pas le cas inversement. En effet, la PI représente un ensemble plus grand et complexe de composantes qui ne sauraient, selon la littérature, se limiter à la PA. Il paraît donc exact de considérer la PA comme étant une partie de la PI, ce qui a d'ailleurs été suggéré explicitement par certains auteurs (Curzon et McOwan, 2017 ; Wing, 2010 ; Tchounikine, 2016).

Bref, la littérature nous a appris que la programmation est une activité en deux volets : l'acte de programmer et l'acte de coder. Ces deux actes renvoient à deux types de processus, le premier étant davantage lié aux algorithmes et à leur conception (se rapprochant davantage de la PA), le second étant plutôt lié à la transposition de ces algorithmes en un langage pouvant être compris et interprété par l'ordinateur (se rapprochant davantage de la PI).

Nous avons tracé le portrait de la PI à l'aide d'une littérature vaste, mais tout de même convergente (tableau 3). Cette convergence s'est exprimée par la forte récurrence des concepts d'abstraction, de résolution de problèmes, de simulations et modèles, puis d'algorithmes (et de la PA), et ce, chez la plupart des auteurs que nous avons cités. En revanche, certaines composantes n'ont été mentionnées qu'à très peu de reprises, voire de façon anecdotique, comme la décomposition ou la pensée logique. Ces composantes ont parfois figuré de façon implicite dans le discours des auteurs, ou ont été intégrées au sein d'autres composantes, ce qui justifie leur présence dans notre tableau synthèse. Nous constatons également une forte cooccurrence de la PI et de la PA. L'explicitation de la PA nous a permis de mettre en évidence les liens qui unissent l'informatique (et la PI) à la mathématique, chose importante considérant la prédominance de la RP et les composantes inhérentes à la programmation (variables, fonctions, etc.). Cela nous paraissait donc essentiel pour bien comprendre la PI et la programmation.

Tableau 3

Synthèse des composantes de la pensée informatique

	Abstraction	Décomposition	Pensée logique	Résolution de problèmes	Généralisation	Automatisation	Simulations et modèles	PA et algorithmes*
Wing, 2006, 2010	x		x	x	x		x	x
Aho, 2012	x			x			x	x
Barr et Stephenson, 2011	x			x		x	x	x
Brennan et Resnick, 2012	x			x		x	x	
Curzon et McOwan, 2017	x	x		x	x		x	x
Delcker et Ifenthaler, 2017	x			x			x	x
Hu, 2011	x		x	x		x	x	x
ISTE et CSTA, 2011	x		x	x	x	x	x	x
Kazimoglu et al., 2012	x			x			x	x
Romero, 2017	x			x			x	x
Tchounikine, 2016	x	x		x	x			x
Voogt et al., 2015	x			x				x

Ce tableau est le résultat d'un processus de sélection subjectif : il permet de démontrer, de façon claire, les principales composantes de la PI selon ces auteurs. Comme nous l'avons mentionné plus tôt, cet article n'est pas le résultat d'un processus systématisé ; il brosse plutôt un portrait compréhensif de la PI et des concepts associés. Les colonnes de ce tableau, exception faite de la dernière*, sont les composantes identifiées par Selby et Woollard (2010) dans leur revue de la littérature. Nous avons rempli le tableau en fonction de notre propre revue de la littérature.

En ce qui concerne la programmation, il s'agit d'une activité complexe nécessitant plusieurs compétences et habiletés. Ces habiletés appartiennent à la PI, qui va toutefois au-delà de cette activité, voire au-delà de l'informatique en soi. En effet, la PI est un ensemble d'habiletés qui peuvent être mobilisées tant en situation de RP avec l'ordinateur que dans un contexte déconnecté (unplugged), sans ordinateur. L'abstraction, la simulation (modélisation) et les algorithmes (et la PA) y sont rattachés. Cette activité complexe peut être effectuée aussi bien à l'école qu'à la maison.

Conclusion

L'objectif de cet article était donc d'offrir un bref aperçu du paysage conceptuel et théorique de la programmation informatique. Comme il a été possible de le constater, bien que la pensée informatique occupe une place importante dans la littérature, il s'avère que sa définition et ses composantes dépendent de nombreux concepts connexes, tels que la pensée algorithmique et la résolution de problèmes. Cette exploration a d'ailleurs permis de mettre en relief les nombreux rapprochements entre la programmation et la mathématique, de même qu'avec la résolution de problèmes. Ce type de réflexion est important dans la mesure où il permet non seulement de nourrir les travaux associés à la programmation et à son rôle, mais également de tisser des liens entre différents champs de recherche. Cela pourrait, nous l'espérons, susciter de nouvelles collaborations interdisciplinaires et ainsi favoriser l'avancement des travaux portant sur la programmation en tant qu'outil d'apprentissage.

Enfin, de façon plus pratique, nous rappelons l'importance de la programmation pour les élèves, non seulement en raison de la démocratisation des outils de programmation, qui permet un accès facile et gratuit à de nombreux logiciels et applications en ligne, mais également parce qu'elle permet aux élèves de mettre en œuvre, voire de développer, un ensemble de compétences et d'habiletés.

Note

[1] Recherche effectuée le 5 avril 2021 sur le site Google Scholar. Les résultats sont approximatifs.

Références

Aho, A. V. (2012). Computation and Computational Thinking. *The Computer Journal, 55*(7), 832-835. https://doi.org/10.1093/comjnl/bxs074

Baron, G.-L. et Bruillard, E. (2001). Une didactique de l'informatique? *Revue française de pédagogie,* (135), 163-172. https://www.jstor.org/stable/41201696

Barr, V. et Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *Acm Inroads, 2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Bossuet, G. (1982). *L'ordinateur à l'école*. Presses Universitaires de France.

Brennan, K. et Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. American Educational Research Association meeting, Vancouver, Canada. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Briant, N. et Bronner, A. (2015, octobre). Étude d'une transposition didactique de l'algorithmique au lycée: une pensée algorithmique comme un versant de la pensée mathématique. Dans L. Theis, *Pluralités culturelles et universalité des mathématiques : enjeux et perspectives pour leur enseignement et leur apprentissage*. Colloque EMF2015 – GT3, Alger, Algérie. http://emf.unige.ch/files/2114/6401/7919/EMF2015GT3BRIANT.pdf

Caspersen, M. E. et Nowack, P. (2013). Computational thinking and practice: A generic approach to computing in Danish high schools. Dans *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136*. 15th Australasian Computing Education Conference, Adelaide, Australie. https://dl.acm.org/doi/proceedings/10.5555/2667199

Curzon, P. et McOwan, P. W. (2017). *The power of computational thinking : games, magic and puzzles to help you become a computational thinker*. World Scientific. https://doi.org/10.1142/q0054

Delcker, J. et Ifenthaler, D. (2017). Computational Thinking as an Interdisciplinary Approach to Computer Science School Curricula: A German Perspective. Dans P. J. Rich et C. B. Hodges (dir.), *Emerging Research, Practice and Policy on Computational Thinking. Educational Communications and Technology: Issues and Innovations* (p. 49-62). Springer. https://doi.org/10.1007/978-3-319-52691-1_4

Futschek, G. et Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. Dans J. E. Clayson et I. Kalas, *Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century*. The 12th EuroLogo conference, Paris, France. http://www.worldcat.org/oclc/700137536

Futschek, G. et Moschitz, J. (2011, octobre). Learning algorithmic thinking with tangible objects eases transition to computer programming. Dans I. Kalaš et R. T. Mittermeir, *Lecture Notes in Computer Science, vol. 7013*. Informatics in Schools. Contributing to 21st Century Education - ISSEP 2011, Bratislava, Slovaquie. https://doi.org/10.1007/978-3-642-24722-4_14

Grover, S. et Pea, R. (2013). Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher, 42*(1), 38-43. https://doi.org/10.3102/0013189X12463051

Guttag, J. V. (2017). *Introduction to Computation and Programming Using Python: With Application to Understanding Data* (2e éd.). The MIT Press.

Hardouin-Mercier, G. (1974). *Technique de la programmation*. Masson.

Henri, F. (2014). *Les bases de la programmation*. JFD éditions.

Hromkovič, J., Kohn, T., Komm, D. et Serafini, G. (2016). Examples of algorithmic thinking in programming education. *Olympiads in Informatics, 10*(1-2), 111-124. https://doi.org/10.15388/ioi.2016.08

Hu, C. (2011, Juin). Computational thinking: what it might mean and what we might do about it. Dans G. Rößling, *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ITiCSE11, Darmstadt, Allemagne. https://doi.org/10.1145/1999747.1999811

ISTE et CSTA. (2011). *Operational Definition of Computational Thinking*. http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf

Kazimoglu, C., Kiernan, M., Bacon, L. et MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science, 9*, 522-531. https://doi.org/10.1016/j.procs.2012.04.056

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly, 92*(3), 170-181. https://doi.org/10.1080/00029890.1985.11971572

L'enseignement de l'informatique à l'école secondaire. (1970, mars). Dans OCDE et Centre pour la Recherche et l'Innovation dans l'enseignement, *Actes du Séminaire pour l'enseignement de l'informatique à l'école secondaire*. Séminaire pour l'enseignement de l'informatique à l'école secondaire, Sèvres, France. http://www.epi.asso.fr/revue/histo/h70ocde.htm

Lamagna, E. A. (2015). Algorithmic thinking unplugged. *Journal of Computing Sciences in Colleges, 30*(6), 45-52. https://dl.acm.org/doi/abs/10.5555/2753024.2753036

Lopez, J. (1986). *Des algorithmes aux langages: basic, LSE, logo*. Hachette.

Lu, J. J. et Fletcher, G. H. L. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin, 41*(1), 260. https://doi.org/10.1145/1539024.1508959

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L. et Settle, A. (2014, juin). Computational thinking in K-9 education. Dans A. C. Clear et R. Lister, *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*. ITiCSE-WGR '14, Uppsala, Suède. https://doi.org/10.1145/2713609.2713610

McMaster, K., Rague, B. et Anderson, N. (2010, décembre). Integrating mathematical thinking, abstract thinking, and computational thinking. Dans *2010 IEEE Frontiers in Education Conference (FIE)*. Frontiers in Education Conference (FIE), Arlington, États-Unis. https://doi.org/10.1109/FIE.2010.5673139

Ministère de l'Éducation et de l'Enseignement supérieur. (2018). *Plan d'action numérique en éducation et en enseignement supérieur*. http://www.education.gouv.qc.ca/fileadmin/site_web/documents/ministere/PAN_Plan_action_VF.pdf

Mirabail, M. (1990). La culture informatique. *ASTER - Recherches en didactique des sciences expérimentales* (11), 11-28. https://www.persee.fr/doc/aster_0297-9373_1990_num_11_1_950

Modeste, S. (2012, février). La pensée algorithmique: apports d'un point de vue extérieur aux mathématiques. Dans J.-L. Dorier et S. Coutat, *Actes du colloque EMF 2012*. Colloque Espace Mathématique Francophone, Genève, Suisse. https://publimath.univ-irem.fr/biblio/ACF12042.htm

Papert, S. (1981). *Jaillissement de l'esprit*. Flammarion.

Parmentier, Y. (2018, avril). Enseigner la pensée informatique à l'école primaire: formation initiale et continue des professeurs. Dans *Atelier «Organisation et suivi des activités d'apprentissage de l'informatique: outils, modèles et expériences»* RJC-EIAH 2018, Besançon, France. https://hal.archives-ouvertes.fr/hal-01762626

Rice, J. K. et Desautels, E. (1969). *Introduction to computer science; problems, algorithms, languages, information and computers*. Holt, Rinehart and Winston.

Romero, M. (2017). Les compétences pour le XXIe siècle. Dans M. Romero, B. Lille et A. Patiño (dir.), *Usages créatifs du numérique pour l'apprentissage au XXIe siècle*. Presses de l'Université du Québec.

Romero, M. (2018, juin). Développer la pensée informatique pour démystifier l'intelligence artificielle. *Bulletin de la société informatique de France,* (12), 67-75. https://doi.org/10.48556/SIF.1024.12.67

Romero, M. et Vallerand, V. (2016). *Guide d'activités technocréatives pour les enfants du 21e siècle*. CoCreaTIC. https://lel.crires.ulaval.ca/sites/lel/files/guidev1._guide_dactivites_technocreatives-romero-vallerand-2016.pdf

Selby, C. et Woollard, J. (2013). *Computational thinking: the developing definition*. University of Southampton (E-prints). https://eprints.soton.ac.uk/356481/

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G. et Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380. https://doi.org/10.1007/s10639-012-9240-x

Serafini, G. (2011, octobre). Teaching programming at primary schools: visions, experiences, and long-term research prospects. Dans I. Kalaš et R. T. Mittermeir, *Lecture Notes in Computer Science vol. 7013*. Informatics in Schools. Contributing to 21st Century Education - ISSEP 2011, Bratislava, Slovaquie. https://doi.org/10.1007/978-3-642-24722-4_13

Tatar, D., Harrison, S., Stewart, M., Frisina, C. et Musaeus, P. (2017). Proto-computational Thinking: The Uncomfortable Underpinnings. Dans P. J. Rich et C. B. Hodges (dir.), *Emerging Research, Practice and Policy on Computational Thinking. Educational Communications and Technology: Issues and Innovations* (p. 49-62). Springer. https://doi.org/10.1007/978-3-319-52691-1_5

Tchounikine, P. (2016). *Initier les élèves à la pensée informatique et à la programmation avec Scratch*. http://codebc.ca/wp-content/uploads/2017/02/PenseeInformatiqueEcole.pdf

Turski, W. M. (1978). *Computer programming methodology*. London.

Van Roy, P. et Haridi, S. (2004). *Concepts, techniques, and models of computer programming*. MIT Press.

Voogt, J., Fisser, P., Good, J., Mishra, P. et Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies, 20*(4), 715-728. https://doi.org/10.1007/s10639-015-9412-6

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. et Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127-147. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. M. (2006, mars). Computational thinking. *Communications of the ACM, 49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2010). Computational Thinking: What and Why? *The Link*. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Pour citer cet article

Parent, S. (2022). Pensée informatique : portrait conceptuel des aspects inhérents à la programmation en contexte scolaire. *Formation et profession, 30*(2), 1-15. https://dx.doi.org/10.18162/fp.2022.651