Didactique de l'informatique : une formation nécessaire

▶ **Christophe DECLERCQ** (LIM, INSPE de l'académie de la Réunion, Université de la Réunion)

■ **RÉSUMÉ** • A partir de l'expérience de l'auteur en formation d'enseignants d'informatique, cette rubrique propose un parcours subjectif parmi les travaux pouvant être référencés et utilisés en didactique de l'informatique pour répondre aux questions professionnelles des enseignants d'informatique au lycée en France. En conclusion des pistes pour la recherche et la formation continue des enseignants sont esquissées.

■ **MOTS-CLÉS** • Didactique de l'informatique, formation des enseignants.

■ ***ABSTRACT*** • *This paper gives some references used in computer science teachers professional development in France. In conclusion, it made proposals for research and the continuing education of teachers.*

■ ***KEYWORDS*** • *Computer science, teacher training.*

1. Introduction

En 1988, Jacques Arsac (Arsac, 1988) situait la didactique de l'informatique comme un problème ouvert et en évoquait une des difficultés fondamentales : « *Il semble à peu près compris que programmer n'est pas résoudre un problème, c'est le faire résoudre par une machine* ». Quelques années et réformes plus tard, Georges Louis Baron et Eric Bruillard (Baron et Bruillard, 2001) interrogeaient encore l'existence de cette didactique dans un contexte de va-et-vient entre une informatique « *objet d'enseignement* » et une informatique « *outil* ». En 2015, Janine Rogalski (Rogalski, 2015) proposait un parcours historique de quelques décennies de travaux en didactique de l'informatique pouvant pour la plupart être encore utilisés aujourd'hui.

Alors que les conditions d'émergence de l'informatique en tant que discipline scolaire avaient été posées dès 1987 par Baron (Baron, 1987), « *un corps d'enseignants professionnels, dont la compétence est garantie par la possession d'un grade, des horaires fixés réglementairement, des programmes nationaux d'enseignement, qui définissent le savoir à enseigner, une Inspection Générale, et des examens finaux* », ce n'est finalement qu'en 2020 que l'ensemble de ces conditions étaient effectivement réunies en France après la création de la spécialité « Numérique et Sciences Informatiques » au lycée et d'une spécialité homonyme du concours de recrutement du CAPES. La formation préparant à ce concours, le master Métiers de l'Enseignement de l'Éducation et de la Formation (MEEF 2nd degré), comportant une part importante de didactique disciplinaire, la question de l'existence d'une didactique de l'informatique ne se pose plus : il s'agit maintenant d'une nécessité. On n'oserait supposer qu'une discipline enseignée uniquement dans l'enseignement supérieur n'a pas besoin de didactique, mais a contrario une discipline scolaire en a une nécessité inscrite réglementairement dans le programme des formations d'enseignants.

La problématique, qui se pose aux formateurs d'enseignants, est alors de rassembler dans un corpus cohérent de « didactique de l'informatique », des travaux publiés, à différentes époques et dans différents contextes, sur l'enseignement de l'informatique. Il n'est pas question ici d'une approche académique visant l'exhaustivité mais plus prosaïquement de rendre accessibles des travaux permettant de répondre à des questions professionnelles de futurs enseignants. Ces questions peuvent porter sur le savoir à enseigner, sur les compétences attendues

des élèves, sur les obstacles à anticiper, les situations didactiques à inventer, les raisons de choisir un environnement d'apprentissage, l'évaluation, ainsi que sur les formes d'enseignement. L'absence, à l'heure actuelle, de cadre de référence unique pour une didactique de l'informatique scolaire (Fluckiger, 2019) incite à emprunter des références aux sciences de l'éducation, aux didactiques voisines, à la psychologie et à l'ergonomie. Cette rubrique propose un parcours de ces références en les situant dans le contexte historique de l'introduction de l'informatique au niveau scolaire en France, et en illustrant l'usage qui peut en être fait pour tenter des réponses aux questions professionnelles des enseignants d'informatique.

1.1. Quelques repères historiques, épistémologiques et curriculaires

L'histoire de l'enseignement de l'informatique en France entre 1970 et 1985 a été relatée par un de ses acteurs, Claude Pair (Pair, 1987). L'émergence d'une communauté francophone de didactique de l'informatique entre 1970 et 2000 a été décrite en particulier par Baron et Bruillard (Baron et Bruillard, 2001). La série de colloques Didapro, qui faisait suite aux colloques organisés par l'association francophone de didactique de l'informatique, a remis à l'ordre du jour les travaux sur l'enseignement de l'informatique en particulier depuis les éditions 2018 (Parriaux *et al.*, 2018) et 2020 (Caron *et al.*, 2020).

Les programmes du lycée publiés en 2018 dans le cadre de la réforme du baccalauréat reposent explicitement sur la description épistémologique de l'informatique proposée par Gilles Dowek (Dowek, 2011) qui en a décrit les quatre piliers - information, algorithme, langage et machine - ainsi que les relations entre ces concepts fondamentaux. Des développements épistémologiques plus récents (Delmas-Rigoutsos, 2018), (Delmas-Rigoutsos, 2020) permettent aussi d'approfondir le caractère fondamental de ces concepts et leurs liens avec les autres disciplines scientifiques.

On remarque que certaines notions méritent d'être reconstruites dans le contexte de l'informatique. C'est le cas en particulier de la notion d'algorithme qui avait, dans le contexte des mathématiques, une définition beaucoup plus restrictive (Modeste *et al.*, 2010) liée au contexte d'usage des algorithmes en mathématiques principalement pour faire du calcul. En informatique, on considère aussi des algorithmes non-

terminants pour décrire des procédés de contrôle ainsi que des algorithmes probabilistes, donc non-déterministes. On s'accorde ainsi pour définir l'algorithme comme la description d'un procédé de traitement ou d'une méthode de calcul ou de résolution de problèmes. On ajoute aussitôt pour éviter toute confusion avec la notion de programme, que cette description est destinée au lecteur humain et qu'elle sera jugée suffisamment précise si on peut l'utiliser pour effectuer la preuve de correction ou l'évaluation de la complexité de cet algorithme.

Une description détaillée des enseignements du lycée figure au bulletin officiel (Bulletin officiel, 2019). Pour la formation des enseignants, on s'intéresse en particulier à l'enseignement de « Sciences numériques et technologie » (SNT) obligatoire en classe de seconde (élèves de 15/16 ans) et à l'enseignement de spécialité « Numérique et sciences informatiques » (NSI) optionnelle en classes de première et terminale (élèves de 16 à 18 ans) du lycée.

Si on se réfère aux spécialistes des curricula qui décrivent l'informatique à la fois comme une science et un ensemble de techniques (Bruillard, 2017), l'enseignement SNT aurait dû être nommé « Science et technologies du numérique », la science (au singulier) du numérique étant l'informatique. C'est d'ailleurs précisément un curriculum d'initiation à l'informatique adossé à une exploration de quelques champs d'usage des technologies du numérique (internet, photographie, objets connectés...). L'enseignement SNT peut ainsi être présenté comme un enseignement d'informatique pour comprendre le monde numérique. Cela permet d'en proposer une didactique fondée sur les pratiques sociales de référence de Jean-Louis Martinand comme le proposait déjà Christian Orange dans les années 1990 (Orange, 1990).

L'enseignement de spécialité « Numérique et sciences informatiques », dont le pluriel intrigue jusqu'à l'Académie des sciences, est un enseignement disciplinaire d'informatique dont la présentation se conforme à l'épistémologie de la discipline. La description du programme inclut principalement une présentation en termes savants des notions et capacités attendues précédée d'un préambule concernant les compétences et les formes d'enseignement, dont le travail par projet. L'enseignant a la responsabilité de mettre en correspondance formes d'enseignement et objectifs en termes de compétences avec les exigences en termes de contenus du programme. On verra quels cadres théoriques peuvent accompagner l'enseignant dans cette voie.

1.2. A la recherche d'un modèle des situations d'enseignement/apprentissage de l'informatique

Selon les auteurs (Fluckiger, 2019), la formalisation d'une situation d'enseignement/apprentissage inclut le savoir, un (ou plusieurs) élèves, un (ou plusieurs) enseignants, des situations d'apprentissage ainsi qu'éventuellement des instruments utilisés pour l'apprentissage. Selon les théories, l'accent est porté principalement sur l'élève, sur le savoir, sur les situations problèmes dont la résolution peut mener à la construction du savoir ou des compétences visées, ou sur l'activité de l'enseignant. Le déroulement des activités d'apprentissage peut s'effectuer de manière synchrone ou asynchrone, les différents acteurs se trouvant en présence ou à distance.

La complexité des situations d'enseignement/apprentissage de l'informatique, outre le fait déjà évoqué qu'il ne s'agit pas de résoudre des problèmes mais de décrire comment une machine pourrait les résoudre, tient aux multiples rôles que peut avoir l'informatique dans ces situations. L'informatique est bien sûr d'abord objet d'enseignement. L'outil informatique peut aussi être un moyen de communication permettant de désynchroniser, mettre à distance, ou faire collaborer les élèves dans leurs apprentissages. L'enseignement de notions informatiques nécessite aussi des pratiques qui requièrent l'usage d'instruments permettant d'exécuter ou interpréter des programmes, des requêtes ou des commandes et d'en visualiser les effets.

Il y a ainsi de multiples sources de quiproquos si l'élève confond l'un et l'autre, ce qui demande de la part de l'enseignant la plus grande rigueur dans la rédaction des consignes. On distinguera aisément : « calcule... », « écrit un programme qui calcule... », « envoie un programme qui calcule... », « écrit un programme qui envoie le résultat du calcul... », « écrit un programme qui écrit un programme qui... ».

On a pu observer (Declercq et Tort, 2018) dès l'initiation à l'informatique, des confusions entre les postures d'utilisateur de télécommande, de programmeur de télécommande et de programmeur. Ces confusions perdurent jusqu'au lycée, quand une même consigne peut être interprétée par l'élève comme une action à décrire dans un langage de programmation ou une interaction avec un environnement de programmation. Cette confusion peut se produire par exemple pour

l'affichage de résultats intermédiaires en cours d'exécution que l'on peut confier à l'environnement ou décrire par des instructions du langage.

Face à cette complexité d'une situation générale d'enseignement/apprentissage de l'informatique, et en l'absence de théorie générale, les formateurs d'enseignants ont besoin de s'appuyer sur des cadres théoriques variés. Les parties suivantes vont s'intéresser successivement aux compétences (partie 2), à l'activité de l'élève (partie 3), aux situations et aux savoirs (partie 4), puis aux instruments indispensables aux apprentissages en informatique (partie 5).

2. Les compétences de la « pensée informatique »

L'expression « *computational thinking* » introduite par Jeannette Wing (Wing, 2006) fait référence à des compétences et des habiletés humaines, pouvant être développées à l'occasion d'activités de programmation, et transférables à d'autres situations de type « résolution de problème ». Il ne s'agit pas de penser comme une machine, mais de décrire les compétences cognitives en jeu pour résoudre, entre autres, un problème informatique, ou plutôt pour le faire résoudre par une machine. Il s'agit d'une activité cognitive de haut niveau et donc bien d'une activité humaine. L'énumération des compétences en jeu fait débat. La traduction française aussi fait débat. « *Thinking* » aurait pu être traduit par « réflexion ». Le terme « computationnel » n'existant pas, l'adjectif « calculatoire » est le plus proche mais semble réducteur.

Dans le contexte de l'apprentissage de la programmation, on utilise les compétences proposées par Selby et Woollard (Selby et Woollard, 2014) pour décrire et qualifier les activités proposées à des élèves. On a reformulé les compétences sous forme verbale de la manière suivante :

– **évaluer** : capacité à attribuer mentalement une valeur (résultat, type...) à un programme donné,

– **anticiper** : capacité à se mettre dans la posture du programmeur qui doit décrire dans un algorithme l'enchaînement séquentiel/répétitif/conditionnel des instructions, avant même le début de son exécution. On préfère ce terme au terme original de « algorithmic thinking » qui encourt le risque de confusion avec « computational thinking »,

– **décomposer :** capacité à transformer un problème complexe en un ensemble de problèmes plus simples équivalents au problème initial,

– **généraliser :** capacité à inférer un problème général à partir d'une instance de ce problème et à repérer dans un problème particulier la répétition de traitements ou de données suivant un même schéma,

– **abstraire :** capacité à « faire abstraction » des informations non pertinentes et à créer des solutions où la manière dont un problème est résolu peut être « abstraite » à l'aide d'une interface pertinente.

Évaluer un programme (lui donner une valeur) peut se faire de différentes manières dans différents domaines. Le plus simple est bien sûr d'évaluer le résultat que donne le programme à l'exécution. Mais il est aussi utile de savoir évaluer le type du résultat sans chercher nécessairement à connaître sa valeur : « *It is interpreting code as data and data as code. It is type checking as the generalization of dimensional analysis*[1] » (Wing, 2006). De manière générale, évaluer un programme consiste donc à regarder ce programme comme une donnée et à en calculer une valeur par une méthode particulière. Ce changement de plan du programmeur consiste à regarder son programme tel qu'il est, et non tel qu'il aurait voulu qu'il soit. La compétence « évaluer » est fondamentale pour mettre au point un programme.

Anticiper est une compétence fondamentale pour la conception d'algorithmes. C'est aussi un des principaux obstacles didactiques rencontrés par les programmeurs débutants : « *Une propriété difficile à intégrer [...] est le caractère différé d'une exécution du programme* » (Rogalski et Samurçay, 1986). C'est la part créative du travail du programmeur d'imaginer par quel chemin de calculs intermédiaires on peut passer, pour aller des informations disponibles aux informations que l'on souhaite calculer. Anticiper les étapes successives du traitement et les différents cas à envisager, c'est tout l'art de programmer.

Décomposer permet d'envisager le traitement de problèmes arbitrairement complexes par réduction à des problèmes plus simples ou déjà connus. La décomposition par cas peut se pratiquer à tous les niveaux de la conception d'un programme : du niveau le plus fin pour séparer quelques instructions au niveau le plus global pour séparer des traitements demandant chacun plusieurs centaines ou milliers de lignes

[1] Traduction : « C'est interpréter un programme comme une donnée et une donnée comme un programme. On considère la vérification de types comme une généralisation de l'analyse dimensionnelle ».

de code. La décomposition séquentielle peut être mise en œuvre avec des variables intermédiaires et des fonctions ou procédures définies pour résoudre les problèmes élémentaires. Le résultat de la démarche de conception peut ainsi être montré explicitement.

Généraliser est une compétence de haut niveau qui permet au programmeur de résoudre des problèmes plus généraux et ensuite de réutiliser pour des instances particulières des parties de programme déjà écrites. La notion de paramètre, utilisée pour instancier des valeurs ou des fonctions particulières est le mécanisme permettant de mettre en œuvre la généralisation. Dans le contexte de la programmation objet, un autre mécanisme permet de prévoir des solutions générales à toute une classe de problèmes, c'est le mécanisme de l'héritage entre classes.

Abstraire avec les données consiste à encapsuler un certain nombre d'informations en utilisant une structure de données composée, qui peut éventuellement masquer le détail du codage des informations. Abstraire avec les fonctions permet de masquer la méthode de calcul utilisée. La combinaison des deux méthodes peut aboutir à la programmation orientée objet où données et méthodes sont encapsulées à l'intérieur d'une classe.

Ces compétences sont citées dans le préambule commun aux programmes de première et de terminale. Leur maîtrise par l'enseignant est nécessaire pour qualifier les activités proposées aux élèves et aussi pour en mesurer la difficulté. Les activités d'évaluation - « quel est le résultat de ... » - sont les plus simples. Les tâches de programmation mobilisent le plus souvent les compétences « anticiper » et « décomposer ». Les compétences « abstraire » et « généraliser » sont plus exigeantes au niveau cognitif car de plus haut niveau.

La maîtrise de ce cadre permet à l'enseignant, à la fois, de s'engager dans une démarche d'évaluation par compétences et de varier les activités proposées aux élèves sur la base de cette typologie.

3. Les apports de la psychologie de la programmation

La psychologie de la programmation (Hoc, 1982) est un domaine de recherche qui a émergé dans le contexte de l'ergonomie et de la didactique professionnelle et s'est consacré à l'étude des situations de travail des programmeurs. Ce n'étaient donc pas, à l'origine, des travaux liés à l'enseignement mais rapidement la question de l'enseignement de méthodes de programmation s'est posée (Rogalski, 1988), (Rogalski *et al.*,

1988). Les difficultés cognitives rencontrées par les programmeurs débutants ont été signalées (Rogalski et Samurçay, 1986).

En particulier, les difficultés de conceptualisation par les élèves de la notion de variable ont été mises en évidence par Renan Samurçay (Samurçay, 1985) :

– *« la construction de la signification et des opérations sur les variables : la déclaration, l'affectation des valeurs, l'entrée et la sortie des données sur l'écran,*

– *le contrôle des valeurs particulières qui doivent être prises par les variables lors de l'exécution du programme : ce contrôle intervient dans la planification des instructions par les structures que sont la répétition, le choix et la séquentialité ».*

La typologie introduite par Samurçay distingue les « variables données », qui sont des données explicites du problème, et ne varient pas, des variables nécessitées par la résolution informatique, avec parmi elles les « variables compteurs », les « variables accumulateurs » et les « variables intermédiaires ». Cette distinction par rôle est utile pour l'enseignant pour graduer les difficultés lors de l'introduction des variables des situations les plus simples (variables données puis compteurs) aux situations les plus générales (variables accumulateurs et variables intermédiaires).

Le rôle précurseur de notions déjà vues par les élèves en mathématiques dans l'assimilation ou l'accommodation de nouvelles notions en informatique a été étudié par Janine Rogalski (Rogalski, 1987) concernant les notions de variable et de fonction : *« La variable a comme précurseur possible la variable mathématique. »* Les précurseurs ont un double rôle : producteur et réducteur. *« Le caractère statique de la variable mathématique peut constituer un obstacle à la représentation de la modification possible de la valeur d'une variable lors de l'exécution d'un programme. [...] L'existence de la représentation symbolique = de l'égalité des variables numériques joue un rôle producteur dans les acquisitions initiales des tests en programmation... mais peut rendre difficile le changement de point de vue qui consiste non pas à comparer... mais à tester si une certaine propriété est vraie ».*

Une synthèse des travaux de ce domaine, qui a été particulièrement actif pendant les années de l'option informatique des lycées, a été publiée récemment (Lagrange et Rogalski, 2017) et reste d'actualité pour anticiper nombre de difficultés dans les apprentissages de l'informatique au lycée.

On retiendra en particulier :

– La difficulté pour les élèves de conceptualiser le calcul booléen puisque « *les élèves conçoivent les expressions booléennes comme des « conditions » telles qu'elles s'expriment dans le langage usuel, plutôt que comme un calcul sur des valeurs logiques »*.

– Le risque de confusion entre égalité et affectation. Leurs notations sont proches et varient selon les langages. En particulier le symbole = peut dénoter l'affectation dans un langage et l'égalité dans un autre.

– La question de « *comment articuler l'élaboration de situations didactiques centrées sur l'acquisition de concepts déterminés (variables, boucles, conditionnelles) avec une activité de programmation partant de problèmes consistants, plus ouverts »* reste vive car les programmes de NSI incitent à une pédagogie par projets tout en étant principalement définis par une listes de notions à étudier.

Dans leur conclusion Rogalski et Lagrange soulignent : « *il apparaît que l'enjeu central est, pour les élèves que nous avons observés, de comprendre la construction d'un programme ou d'un algorithme comme l'organisation d'un traitement sur un dispositif ; ils doivent percevoir ce dispositif comme un ensemble de variables, et concevoir ces variables comme des objets « calculables » et le traitement comme l'évolution de leurs valeurs »*.

Dans le contexte actuel de la programmation en Python au lycée, il convient aussi le moment venu de passer d'un modèle de dispositif basé uniquement sur les variables en mémoire, à un modèle incluant l'environnement et la mémoire. Cette étape est nécessaire pour bien comprendre les données mutables et les mécanismes de passage de paramètres. Cette dernière question ne peut être éludée car elle est la source de questions professionnelles pour l'enseignant.

4. Des cadres de référence empruntés aux didactiques des mathématiques

Dans l'étude des situations d'apprentissage proposées aux élèves, c'est sans surprise la théorie des situations didactiques de Guy Brousseau (Brousseau, 1998), empruntée à la didactique des mathématiques, qui semble la plus utilisée pour analyser les activités. Les notions de milieu, de variable didactique, de situation, d'obstacle sont indispensables pour analyser a priori les activités proposées aux élèves. En particulier, la maîtrise des variables didactiques d'une situation permet à l'enseignant de différentier les activités proposées selon les capacités des élèves. Une communication récente (Meyer et Modeste, 2020), qui propose une

situation fondamentale pour l'étude de la complexité des algorithmes, s'y réfère explicitement et cible des apprentissages au programme de la spécialité NSI. La recherche a permis de confirmer l'hypothèse des auteurs concernant l'intérêt des jeux à deux joueurs pour aborder dans le cadre scolaire la notion de complexité.

De nombreux autres travaux restent à mener pour proposer des situations couvrant l'ensemble des contenus au programme de la spécialité NSI.

C'est aussi aux didactiques des mathématiques que l'on doit l'emprunt de la théorie anthropologique du didactique de Yves Chevallard (Chevallard, 1991). Cette approche a été explorée par Fabienne Viallet et Patrice Venturini dans une étude sur la transposition de la notion de boucle (Viallet et Venturini, 2010). La distinction entre savoir savant et savoir enseigné est particulièrement féconde pour l'enseignement de l'informatique dans un contexte où la plus grande partie des savoirs à enseigner ne l'ont été auparavant que dans l'enseignement supérieur, ce qui explique que le travail de transposition didactique n'ait pas encore été accompli. Ceci est particulièrement flagrant dans le domaine de l'informatique théorique où des enseignants de terminale s'interrogent sur la nécessité d'introduire les machines de Turing pour aborder l'indécidabilité du problème de l'arrêt. Une proposition de preuve de l'indécidabilité du problème de l'arrêt transposée au lycée a été proposée par Journault *et al.* (Journault *et al.*, 2020). C'est aussi prégnant dans le domaine des réseaux informatiques où les filières d'enseignement supérieur délivrent une formation professionnelle alors que les objectifs des programmes de lycées se limitent à une compréhension des technologies employées et de leurs enjeux. L'accompagnement d'enseignants débutants nécessite leur prise de conscience que cette transposition est indispensable pour construire, dans chaque domaine, un savoir à enseigner au lycée à partir du savoir savant qu'ils ont rencontré à l'université.

Cette proximité entre les mathématiques et l'informatique marquée par des emprunts de théories didactiques ne doit cependant pas occulter l'émancipation de l'informatique par rapport aux mathématiques en tant que discipline. Ce processus d'émancipation achevé depuis plus de trente ans à l'université, est encore en cours dans l'enseignement scolaire, sachant qu'une partie des enseignements d'informatique au collège et en classe de seconde est confiée aux professeurs de mathématiques ou de

technologie. De très nombreux enseignants de mathématiques, formés en formation continue via le diplôme inter-universitaire « Enseigner l'informatique au lycée » (Marquet et Declercq, 2019), enseignent d'ailleurs les deux disciplines et doivent à ce titre jongler avec les injonctions contradictoires concernant la manière d'organiser les apprentissages pour la conception d'algorithmes. Une anecdote significative de cette difficulté mérite d'être signalée : une directive concernant l'écriture de programmes à l'épreuve de mathématiques du baccalauréat proscrivant l'usage du *print* et encourageant le *return*, a été interprétée par des enseignants comme s'appliquant aussi à l'informatique. Ceci n'a évidemment pas de sens car la programmation de systèmes interactifs ou réactifs ne peut s'envisager sans l'utilisation d'entrées-sorties.

Le partage de cadres théoriques permet alors d'expliquer pourquoi telle situation est plus propice dans tel milieu et pourquoi le savoir enseigné concernant les algorithmes peut différer en mathématiques et en informatique.

5. L'analyse instrumentale et les environnements interactifs pour l'apprentissage humain

Les apports de la théorie instrumentale de Pierre Rabardel (Rabardel, 1995) ont été résumés dans (Nijimbere, 2013). Conçu initialement pour comprendre l'usage d'instruments dans le travail humain, ces recherches sont particulièrement fécondes aussi pour comprendre l'usage d'instruments dans le cadre d'une activité d'apprentissage.

La compréhension de phénomène d'appropriation d'un artefact pour en faire un instrument (genèse instrumentale), processus dirigé à la fois vers le sujet qui modifie ses schèmes d'action (instrumentation) et vers l'artefact qui est alors adapté à l'action du sujet (instrumentalisation), est utile pour comprendre comment un élève, ou un enseignant, s'approprie par exemple un environnement de programmation. Cela permet de relativiser la perception que l'utilisateur a d'un instrument, perception d'autant plus positive que sa genèse instrumentale est avancée et de mieux comprendre pourquoi le meilleur instrument pour le professeur n'est pas forcément le meilleur pour l'élève.

Pour objectiver les raisons du choix d'un instrument à proposer à l'élève, la notion de « retour instrumental » est fondamentale. Quel retour

l'instrument procure-t-il à l'élève lors de son usage ? Ce retour peut-il le guider dans l'apprentissage ?

Appliqué au choix d'un environnement de programmation, les questions de retour instrumental à se poser sont les suivantes :

– Quel retour lors de l'édition : colorisation syntaxique, indentation automatique, parenthésage automatique ? Ce retour aide-t-il à l'écriture d'un programme correct syntaxiquement ?

– Quel retour avant, ou lors de l'exécution, concernant la correction du programme : problèmes liés au typage, effets de bord non désirés, débordements dans les tableaux, instruction conditionnelle non exhaustive, choix des inégalités, comparaisons et calculs entre flottants, mauvais nommage des variables ? Ces retours sont-ils utilisables par l'élève ?

– Quel retour à l'exécution : distinction *print/return*, exécution pas à pas au niveau instruction ou au niveau expression, visualisation de la mémoire, de l'environnement, de la pile ?

Une étude attentive permet de choisir un environnement en fonction de ce qu'il donne à voir à l'élève des programmes et de leur exécution. Ce choix peut bien sur différer de la seconde à la terminale. En particulier, le modèle de machine que l'environnement laisse entrevoir peut évoluer de la « machine qui associe des valeurs à des variables » à celle qui « montre l'environnement associant des références aux noms, et la mémoire associant des valeurs aux références ».

Pour les premiers apprentissages en programmation, on a proposé un environnement dédié dont les retours instrumentaux garantissent des programmes corrects par construction (Declercq et Nény, 2020).

L'analyse des retours instrumentaux gagnerait aussi à être menée à propos de l'enseignement des systèmes d'exploitation : en effet selon les choix retenus (un simulateur de console ou la console du système) la nature des retours, différés ou non, visuels dans l'explorateur de fichier ou textuel, est très différente.

L'analyse instrumentale est depuis longtemps utilisée dans la communauté qui s'intéresse aux environnements interactifs pour l'apprentissage humain. Ses résultats ne sont pas spécifiques à l'apprentissage de l'informatique mais lui sont applicables en prenant soin de distinguer l'informatique moyen et/ou objet d'enseignement. En particulier les travaux sur les jeux sérieux, les exerciseurs, les

environnements personnels d'apprentissage ou ceux permettant la collaboration sont applicables même s'ils ne sont pas spécifiques et ne sont pas à ce titre répertoriés en didactique de l'informatique.

Des travaux récents sur la collaboration en apprentissage de l'informatique utilisant soit des systèmes issus de l'industrie du logiciel (Raclet *et al.*, 2020) soit des systèmes construits à cette fin (Broisin *et al.*, 2015) relatent des expérimentations dans l'enseignement supérieur. Leur transposition au lycée serait à étudier, vu l'importance accordée aux projets parmi les formes d'enseignement de l'informatique au lycée, et vu la difficulté de gérer la conduite et la documentation d'un projet sans la médiation d'instruments adaptés.

L'analyse instrumentale, même si cela est parfois oublié, s'applique aussi aux instruments tangibles en bois ou en papier, développés dans le cadre du courant dit de l'informatique débranchée (*computer science unplugged*) ou informatique sans ordinateur. Une revue de littérature (Drot-Delange, 2013) a été publiée qui recense de nombreux travaux et les bénéfices de cette approche pour les apprentissages en informatique. Destinée à l'origine plutôt à l'enseignement primaire, l'informatique débranchée a conquis l'enseignement secondaire et est explicitement citée comme modalité d'enseignement dans les programmes de première et de terminale.

Les situations issues de concours telles celles du concours Castor (Drot-Delange et Tort, 2018) constituent aussi une forme originale de situations instrumentées où la résolution de défis peut être utilisée pour construire des apprentissages en informatique.

Dans la diversité des situations évoquées, l'analyse instrumentale est ainsi un cadre indispensable pour l'enseignant qui veut fonder ses choix d'environnements à proposer aux élèves, non sur ses préférences personnelles, mais sur une analyse rationnelle des processus cognitifs en jeu.

6. En guise de conclusion, des pistes pour la recherche et la formation

La présente recension de travaux en didactique de l'informatique ne prétend pas à l'exhaustivité : elle a été construite à partir des références réellement utilisées en formation d'enseignants d'informatique en partie dans le cadre du DIU et surtout dans celui du parcours informatique du master MEEF de l'INSPE de l'académie de Nantes. Cette analyse emprunte

aussi, sans pouvoir encore les citer, à des travaux en cours d'étudiants préparant leur mémoire de master et peut également fournir des références et des idées de problématiques aux promotions suivantes. C'est en particulier à cause de son usage en enseignement que la littérature francophone a été presque exclusivement utilisée.

Pour constituer un corpus plus complet, il conviendrait maintenant de comparer au niveau national les références utilisées en formation d'enseignants d'informatique, et aussi au niveau de la communauté francophone, les hautes écoles pédagogiques en Suisse ou en Belgique ayant déjà une longue expérience de formation d'enseignants du secondaire dans un contexte un peu différent.

On peut aussi formuler l'hypothèse qu'une demande de formation en didactique de l'informatique est en cours d'émergence parmi les enseignants d'informatique au lycée dont la formation par le DIU a été, faute de temps, très majoritairement disciplinaire. C'est à ce public que cette rubrique est dédiée en souhaitant qu'elle suscitera des envies de formation voire de participation à des groupes de recherche-action comme le proposent de nombreux IREM (Institut de recherche en Enseignement des Mathématiques) dont certains deviennent des IREMI (Institut de recherche en Enseignement des Mathématiques et de l'Informatique) incluant l'informatique dans leurs préoccupations.

Gageons aussi que la communauté de recherche francophone en didactique de l'informatique va se renforcer fortement, maintenant que son objet d'étude est devenu une discipline scolaire à part entière. Des signes en attestent déjà avec la constitution d'un groupe de travail sur l'enseignement de l'informatique de la maternelle à l'université (Broisin *et al.*, 2021) au sein de l'ATIEF et la parution de ce numéro spécial.

La prochaine conférence francophone de didactique de l'informatique « Didapro 9 » qui va avoir lieu en mai 2022 à l'INSPE de l'académie de Nantes, présentera de nombreuses contributions sur ces thèmes.

RÉFÉRENCES

Arsac, J. (1988). La didactique de l'informatique : Un problème ouvert ? Dans *Actes du Colloque francophone sur la didactique de l'informatique* (p. 9-18).

Baron, G.-L. (1987). *La constitution de l'informatique comme discipline scolaire, le cas des lycées* [thèse de doctorat, Université Paris Descartes].

Baron, G.-L. et Bruillard, E. (2001). Une didactique de l'informatique ? *Revue Française de Pédagogie*, *135*, 163-172.

Broisin, J., Declercq, C., Fluckiger, C., Parmentier, Y., Peter, Y et Secq, Y. (2021). Dans *Actes de l'atelier APIMU 2021 @ EIAH : Apprendre la Pensée Informatique de la Maternelle à l'Université*. HAL.

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

Brousseau, G. (1998). *Théorie des situations didactiques*. La pensée Sauvage.

Bruillard, E. (2017). Enseignement de l'informatique entre science et usages créatifs : Quelle scolarisation ? Dans H. J. Nguyen et E. Vandeput (.dir), *L'informatique et le numérique dans la classe. Qui, quoi, comment ?* Presses universitaires de Namur.

Bulletin officiel (2019). Spécial n°1 du 22 janvier.

Caron, P.-A., Fluckiger, C., Marquet, P., Peter, Y. et Secq, Y. (2020). *L'informatique, objets d'enseignements enjeux épistémologiques, didactiques et de formation* [communication]. Colloque Didapro 8 – DidaSTIC L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation, Lille, France.

Chevallard, Y. (1991). *La transposition didactique, du savoir savant au savoir enseigné*. La Pensée Sauvage.

Declercq, C et Nény, F. (2020). *Block2Py, un éditeur de blocs pour l'apprentissage du langage Python* [communication]. Colloque Didapro 8 – DidaSTIC L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation, Lille, France.

Declercq, C. et Tort, F. (2018). Organiser l'apprentissage de la programmation au cycle 3 avec des activités guidées et/ou créatives. Dans *Actes des Ateliers RJC EIAH 2018*. https://wikis.univ-lille.fr/computational-teaching/_media/wiki/actions/2018/rjceiah/christophe-declercq-francoise-tort.pdf

Delmas-Rigoutsos, Y. (2020). *Variables, grandeurs et types* [communication]. Colloque Didapro 8 – DidaSTIC L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation, Lille, France.

Chevallard, Y. (1991). *La transposition didactique, du savoir savant au savoir enseigné*. La pensée Sauvage

Delmas-Rigoutsos, Y. (2018). *Proposition de structuration historique des concepts de la pensée informatique fondamentale* [communication]. Colloque Didapro 8 – DidaSTIC L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation, Lille, France.

Dowek, G. (2011). Les quatre concepts de l'informatique. Dans G.-L. Baron, E. Bruillard, et V. Komis (dir.), *Sciences et technologies de l'information et de la communication en milieu éducatif : Analyse de pratiques et enjeux didactiques.* (p. 21-29). New Technologies Editions.

Drot-Delange, B. (2013). Enseigner l'informatique débranchée : Analyse didactique d'activités. *AREF*, 1-13.

Drot-Delange, B. et Tort, F. (2018). *Concours Castor, ressource pédagogique pour l'enseignement de l'informatique ? Étude exploratoire auprès d'enseignants* [communication]. Didapro 7 – DidaSTIC. De 0 à 1 ou l'heure de l'informatique à l'école, Lausanne, Suisse.

Fluckiger. C. (2019). *Une approche didactique de l'informatique scolaire*. Presses universitaires de Rennes.

Hoc, J. M. (1982). L'étude psychologique de l'activité de programmation : Une revue de la question. *Technique et Science Informatique*, *38*(2-3), 213-221.

Journault, M., Lafourcade, P., More, M. et Poulain, R. (2020). *Une preuve pour le lycée de l'indécidabilité du problème de l'arrêt* [communication]. Colloque Didapro 8 – DidaSTIC L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation, Lille, France.

Lagrange, J.-B. et Rogalski, J. (2017). Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique. *Annales de Didactiques et de Sciences Cognitives*, *22*, 119-158.

Marquet, P. et Declercq, C. (2019). DIU Enseigner l'informatique au lycée. *Bulletin de la société informatique de France*, *1024, 47-56*

Meyer, A. et Modeste, S. (2020, février). *Analyse didactique d'un jeu de recherche : Vers une situation fondamentale pour la complexité d'algorithmes et de problèmes* [communication]. Colloque Didapro 8 – DidaSTIC L'informatique, objets d'enseignements – enjeux épistémologiques, didactiques et de formation, Lille, France.

Modeste, S., Gravier, S. et Ouvrier-Buffet, C. (2010). Algorithmique et apprentissage de la preuve. *Repères IREM*, *79*, 51-72.

Nijimbere, C. (2013). Approche instrumentale et didactiques : Apports de Pierre Rabardel. *Adjectif.net*. https://adjectif.net/spip/spip.php?article202

Orange, C. (1990). Didactique de l'informatique et pratiques sociales de référence. *Bulletin de l'EPI*, *60*.

Pair, C. (1987). Informatique et enseignement = hier, aujourd'hui et demain. *Bulletin de l'EPI*, *47*, 85-97.

Parriaux, G., Pellet, J.-P., Baron, G.-L., Bruillard, É et Komis, V. (2018). De 0 à 1 ou l'heure de l'informatique à l'école. Dans *Actes du colloque Didapro 7 – DidaSTIC*. Peter Lang

Rabardel, P. (1995). *Les hommes et les technologies; approche cognitive des instruments contemporains*. Armand Colin.

Raclet, J.-B., Silvestre, F. et Pons, M. (2020). Mise en œuvre d'approches pédagogiques fondées sur des pratiques de l'industrie du logiciel pour l'apprentissage de la programmation. Dans *Actes du Colloque Didapro 8 – DidaSTIC : L'informatique, objets d'enseignements* (p. 1-12).

Rogalski, J. (1987). Acquisition de savoirs et de savoir-faire en informatique. *Cahier de didactique des mathématiques, 43*. IREM Paris VII.

Rogalski, J. (1988). Enseignement de méthodes de programmation dans l'initiation à l'informatique. Dans G.-L. Baron, J. Baudé et P. Cornu (dir.), *Colloque francophone sur la didactique de l'informatique* (p. 61-72). Association EPI.

Rogalski, J. (2015). Psychologie de la programmation, didactique de l'informatique. Déjà une histoire… *Informatique en éducation : perspectives curriculaires et didactiques*.

Rogalski, J. et Samurçay, R. (1986). Les problèmes cognitifs rencontrés par des élèves de l'enseignement secondaire dans l'apprentissage de l'informatique. *European Journal of Psychology of Education*, *1*(2), 97-110.

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

Samurçay, R. (1985). Signification et fonctionnement du concept de variable informatique chez des élèves débutants. *Educational Studies in Mathematics, 16,* 143–161.

Selby, C. et Woollard, J. (2014). Computational thinking: The developing definition. Dans *Actes du SIGCSE*. https://people.cs.vt.edu/˜kafura/CS6604/Papers/CT-Developing-Definition.pdf

Viallet, F et Venturini, P. (2010). *Didactique comparée et enseignement de l'informatique* [communication]. Congrès International Actualité de la Recherche en Éducation et en Formation (AREF), Borgeaux, France.

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