Quelle actualité pour les questions abordées lors du premier colloque de didactique de l'informatique (1988) ?

Monique Grandbastien[1]

[1] LORIA, Université de Lorraine, Nancy, France
monique.grandbastien@loria.fr

Résumé. Ce document présente brièvement les sujets développés lors du premier colloque francophone de didactique de l'informatique (1988) et décrit plus particulièrement ceux relatifs à l'apprentissage des bases de la programmation par des publics débutants, notamment dans l'enseignement secondaire. Il les complète par l'analyse de quelques articles de publications internationales plus récentes (1995-2019). L'objectif est de questionner l'actualité de ces sujets dans le contexte actuel de la création des enseignements de SNT et NSI dans les lycées français pour la formation des maîtres et pour l'observation et la compréhension des difficultés rencontrées par les élèves.

Mots-clés : algorithmique, programmation, informatique au niveau scolaire, didactique de l'informatique.

1 Introduction

En 1988 avait lieu à Paris le premier colloque francophone de didactique de l'informatique. Il était porté principalement par des enseignants d'informatique à l'université et acteurs de formation et d'accompagnement des maîtres pour l'option informatique des lycées à partir de 1981. Il réunissait à la fois des universitaires ayant l'expérience de la formation d'étudiants et de futurs professionnels aux bases de l'informatique et des enseignants de terrain des premier et second degrés, soucieux d'échanger leurs expériences pédagogiques. Les questions de didactique abordées dans les communications de ce colloque sont donc à analyser dans ce contexte pédagogique et dans le contexte technologique de l'époque. J. Arsac, son président, avait intitulé sa communication "la didactique de l'informatique : un problème ouvert ?" ; il semble que trente ans après le problème soit encore plus ouvert.

En effet, en 2019, la communauté des enseignants d'informatique en France (et dans quelques autres pays) retrouve une situation similaire dans un contexte évidemment très différent. Un enseignement de Sciences Numériques et Technologies (SNT) est introduit au lycée dans toutes les classes de seconde, après 20 ans d'interruption. Des milliers de professeurs doivent être formés, des approches pédagogiques sont à inventer, à tester et à diffuser.

Dans ce document, notre objectif est de présenter brièvement les sujets développés lors de ce premier colloque et d'analyser plus particulièrement ce qui concerne

l'apprentissage des bases de la programmation par des publics débutants, notamment dans l'enseignement secondaire. Nous les complétons en interrogeant des publications internationales plus récentes (1995-2019), puis nous questionnons leur actualité dans le contexte actuel. : Y-a-t-il des éléments à reprendre dans ces travaux, notamment pour la formation des maîtres et l'observation et la compréhension des difficultés rencontrées par les élèves actuels ?

2 L'existant en 1988

A la fin des années 80, la programmation était enseignée dans la plupart des universités, parfois depuis plus de vingt ans, et dans certains établissements secondaires, à titre le plus souvent optionnel ou expérimental. Les enseignants avaient très tôt noté les difficultés rencontrées par leurs étudiants et proposé, au fur et à mesure que les fondements de la science informatique étaient explicités [Dijkstra, 1976], [Gries, 1981], [Wirth, 1976], des démarches et des outils pour y remédier. Certaines communications des congrès WCCE[1] de l'IFIP[2] à partir de 1970 et de nombreux ouvrages au niveau français consacrés à la construction de programmes en témoignent comme par exemple [Arsac, 1970], [Arsac, 1980], [Arsac, 1987] et [Ducrin, 1984]. Ces ouvrages sont illustrés par de nombreux exemples dont la résolution est détaillée pas à pas. Les thèmes peuvent faire sourire par rapport aux préoccupations des étudiants actuels, ils n'intègrent pas les approches de construction d'applications par réutilisation de composants, mais le souci du détail dans la description de plusieurs solutions et des raisonnements qui ont conduit à ces solutions mériteraient d'être repris. Les principales questions soulevées par ces premiers cours d'informatique dans l'enseignement supérieur concernaient la créativité à mettre en œuvre dans la résolution des problèmes, la rigueur de l'expression d'une solution et la méthode, organisation rigoureuse du processus conduisant de l'énoncé d'un problème à l'expression d'un programme pour le résoudre.

Au niveau de l'enseignement scolaire, une option informatique des lycées avait été créée en France en 1981 et avait donné lieu à des publications pour faire circuler les expériences entre les acteurs des différentes académies. Une note de service de la direction des lycées (DLC)[3] en 1988 rappelle les brochures éditées par la direction des Lycées et le CRDP[4] de Poitiers « Enseigner l'informatique » (1985), « L'option informatique, réalités et pratiques » (1986), « Pratiques et savoir-faire des élèves de l'option informatique » (1987), « Technologie des ordinateurs et enseignement de l'option informatique » (1987), ainsi que la revue *Informatiques* éditée par les mêmes acteurs. C'est donc dans ce contexte et avec cet existant en mémoire qu'il faut analyser les actes du colloque de 1988.

[1] WCCE : World Congress on Computers in Education
[2] IFIP : International Federation for Information Processing
[3] Direction des Lycées et Collèges, note 88-084, Ministère de l'Education nationale
[4] CRDP : Centre Régional de Documentation Pédagogique

3 Difficultés pédagogiques abordées dans les actes du colloque

Les actes [Baron et al, 1988] contiennent 18 communications dont les titres, fournis dans le tableau 1, montrent une grande diversité d'intérêts. Certaines communications traitent plutôt des objectifs et avantages espérés d'un enseignement d'informatique (3, 4, 13, 15), d'autres de mise en œuvre dans l'enseignement supérieur (11, 17) ou des contraintes propres à l'école primaire (10, 12), d'exemples d'outils logiciels ou de mise en œuvre de démarches particulières (5, 8, 9, 10, 12, 16, 18). Nous analysons principalement dans les paragraphes qui suivent celles qui traitent des questions générales de l'enseignement de la programmation aux débutants (1, 6, 7, 14).

Tableau 1. Sommaire des actes du colloque de 1988

(1) Arsac J.	La didactique de l'informatique : Un problème ouvert
(2) Neuville Y.	L'espace informatique francophone
(3) De Landsheere G.	Processus d'apprentissage et d'enseignement de l'informatique : Quelques questions
(4) Morin B. A.	Pygmalion dans la classe informatique
(5) Guéraud V., Peyrin J.-P.	Un jeu de rôle pour l'enseignement de la programmation
(6) Rogalski J.	Enseignement de méthodes de programmation dans l'initiation à l'informatique
(7) Pair C.	L'apprentissage de la programmation
(8) Deveaux D., Raphalen M., Revault J.	Spécification d'un interprêteur de mini-langage algorithmique pour l'initiation à la programmation
(9) Dufourd G., Dufourd J.-F.	Des univers et des outils variés pour commencer à programmer
(10) Parmentier C.	Didactique et programmation à l'école
(11) Deroite M., Le Charlier B.	Un système d'aide à l'enseignement d'une méthode de programmation
(12) Aigle M.	« Vous avez dit binaire ? »
(13) Cloutier J.-F.	Apport de différents paradigmes de programmation comme autant d'outils de pensée
(14) Hauglustaine –Charlier B.	Images pour apprendre à programmer
(15) Romainville M.	Une analyse critique de l'initiation à l'informatique : quels apprentissages et quels transferts ?
(16) Geoffroy C.	Une initiation à l'analyse descendante
(17) Delacharlerie A.	Apprentissage de la conception des bases de données : une méthodologie de la méthode
(18) Favre-Nicolin R.	Le tableur et l'option informatique : vers la programmation par objets

Une difficulté générale déjà mentionnée dans [Arsac, 1987] tient au fait que l'enseignement de l'informatique, et notamment de la programmation, en dehors du cadre professionnel, n'a pas d'abord pour objectif la transmission de connaissances et de techniques, mais plutôt le développement d'aptitudes intellectuelles, créativité, rigueur de pensée, clarté et précision de l'expression, sens de l'organisation. Pour atteindre ces buts et dans la continuité des sujets débattus dans les ouvrages déjà publiés, certaines communications proposent plutôt des principes illustrés par des exemples, mettant l'accent sur la créativité sans oublier la rigueur, d'autres des méthodes plus strictes dont il ne faudra pas rester prisonnier pour se montrer créatif face à des situations nouvelles. Nous présentons successivement la construction des algorithmes, les méthodes et la phase de programmation-exécution en 3.1, 3.2, 3.3. Un

autre sujet développé en 3.4 concerne les observations relatives aux étudiants. Les auteurs mentionnent que les cours font souvent l'hypothèse d'un public assez homogène, motivé et capable d'autonomie, ce qui est déjà loin d'être toujours le cas à l'Université. Mais pour des élèves plus jeunes, la prise en compte de leurs caractéristiques en termes d'états cognitifs et de prérequis s'impose.

Globalement, ces communications portent davantage sur des réflexions quant aux objectifs et des propositions de mise en œuvre que sur des retours d'expérience avec description des succès et des difficultés observés chez les élèves, ce qui n'est pas surprenant dans une phase de création d'enseignement.

3.1 Du problème à un algorithme

Il est habituel de décomposer l'activité désignée de façon trop générique par le terme « programmation » en plusieurs phases. La première consiste à passer de l'énoncé d'un problème en langue naturelle à la description d'un mode de résolution exprimé comme combinaisons de quelques actions simples. Cette étape est signalée comme difficile par beaucoup d'auteurs, notamment à cause de la distance entre l'énoncé initial d'un problème et un algorithme pour le résoudre.

Parmi les solutions envisagées pour assister l'apprenti programmeur à cette étape, plusieurs auteurs proposent des principes dont on peut s'inspirer alors que d'autres tentent des méthodes plus systématiques dont nous aurons un exemple au paragraphe suivant. Parmi les principes souvent évoqués, on trouve le recours à l'observation d'un humain en train de résoudre un problème analogue, avec plusieurs objections immédiates : il s'agit de faire faire le travail à une machine et le mode opératoire humain n'est pas toujours celui qu'il faut retenir, ensuite un mode opératoire humain peut être difficile à expliciter avec la rigueur et la précision requises par les descriptions d'algorithmes, enfin on cherche en général un algorithme pour le réutiliser plusieurs fois pour la résolution de plusieurs problèmes d'une même famille, ce qui crée une situation différente de la résolution d'un seul problème.

Après la difficulté relative à la découverte d'une stratégie de résolution, les auteurs pointent des difficultés dues aux constructions et aux notations des langages algorithmiques, notamment une qui vient heurter les habitudes prises par les élèves en mathématiques. Il s'agit de la notion de variable, avec, dans les notations mathématiques une association statique entre un nom de variable et une valeur et en informatique une association dynamique, un même nom de variable venant repérer plusieurs valeurs au cours de l'exécution du processus décrit. Cette difficulté a été travaillée en collaboration avec des didacticiens des mathématiques [Nguyen, 2005] dans les IREM[5]s et avec des spécialistes de psychologie cognitive pour la prise en compte des cadres cognitifs préexistant chez les élèves [Samurçay, 1985].

On note enfin l'absence de distinction dans cette phase entre la modélisation des données et celle de résolution du problème.

[5] Institut de Recherche sur l'Enseignement des Mathématiques, http://www.univ-irem.fr/

3.2 La question des méthodes

Avec la question des méthodes de programmation, nous avons un bon exemple de l'imbrication constante entre les progrès de la structuration de la science informatique et leurs conséquences sur l'évolution des enseignements. Faut-il des méthodes et pourquoi ? Un premier objectif tout à fait légitime consiste à assister les élèves dans leur activité de construction de programmes.
L'équipe de C. Pair à Nancy publie, sous le nom collectif Amédée Ducrin, en 1984, un ouvrage d'enseignement de la programmation en deux tomes intitulés respectivement « du problème à l'algorithme » et « de l'algorithme au programme BASIC – PASCAL – LSE ». Le premier tome est entièrement consacré à l'enseignement d'une méthode de construction d'algorithme, la méthode déductive MEDEE, qui guide le programmeur de l'énoncé d'un problème à un algorithme exprimé dans un pseudo-code et permettant sa résolution

Un article [Langer, 2010] du bulletin vert de l 'APMEP[6] publié en 2010 (donc avec des programmes de mathématiques antérieurs aux programmes actuels) propose d'utiliser la méthode MEDEE pour guider les élèves dans la construction d'un algorithme pour réduire des fractions au même dénominateur. Les traductions de cet algorithme sont ensuite proposées en Algobox, Python et Scratch. Il n'y a malheureusement pas de commentaires sur l'activité des élèves, mais a priori le professeur a souhaité proposer une méthode. Nous reproduisons en figure 1 la solution proposée qui nous permet d'illustrer sur cet exemple la méthode MEDEE :

- Partir du résultat final (c'est une méthode descendante) parce que le résultat est en général bien défini dans l'énoncé, donc ici écrire les valeurs calculées pour le numérateur NS et le dénominateur DS (on notera la présence d'un lexique précis à créer en colonne de gauche).
- Le définir dans le pseudo-code proposé en introduisant si nécessaire des intermédiaires et donc en décomposant le problème en autant de sous-
- problèmes, donc ici définir les deux résultats. D'abord NS à partir de N et du PGCD (N,D) dont le calcul nécessite l'introduction d'une fonction, à définir comme module séparé selon la même méthode, puis DS dont le calcul nécessite la valeur du PPCM (D1, D2), D1 et D2 étant les dénominateurs des fractions données.
- Définir de la même façon chaque résultat intermédiaire, donc ici N1, D1, N2, D2 qui sont les données du problème, puis les modules introduits, c'est à dire les fonctions PPCM et PGCD
- S'arrêter quand il n'y a plus de résultat à définir. L'algorithme est l'ensemble des définitions obtenues, à ordonner ensuite pour exécution.

Cependant, après plusieurs années d'expériences et de réflexions, C. Pair [Pair, 1988] se demandait dès 1988, comme d'autres auteurs, si l'on peut vraiment enseigner des méthodes et suggère de mettre simplement les élèves en situations d'en acquérir par leur pratique.

[6] APMEP Association des professeurs de mathématiques de l'enseignement public

Lexique

NS : entier	Numérateur de la somme simplifiée
DS : entier	Dénominateur de la somme simplifiée
N : entier	Numérateur de la somme
P : entier	Plus Grand Commun Diviseur de N et D
D : entier	Plus Petit Commun Multiple de D1 et D2
N1 : entier	Numérateur de la fraction 1
D1 : entier	Dénominateur de la fraction 1
N2 : entier	Numérateur de la fraction 2
D2 : entier	Dénominateur de la fraction 2
PGCD : fonction	Retourne le PGCD de deux entiers.
PPCM : fonction	Retourne le PPCM de deux entiers.

Définitions

10 *résultat = écrire NS, DS*
9 *NS = N÷P*
8 *DS = D÷P*
6 *N = N1*D÷D1 + N2*D÷D2*
7 *P = PGCD(N,D)*
5 *D = PPCM(D1,D2)*
1 *N1 = donnée*
2 *D1 = donnée*
3 *N2 = donnée*
4 *D2 = donnée*

Fonction *PPCM*(a,b)

a : entier	Premier argument de la fonction
b : entier	Deuxième argument de la fonction
m : entier	élément générique de la liste des multiples de a

3 *résultat* = m_i
2 *répéter tant que m mod b* $\neq 0$
m = @m + a
1 $m_1 = a$

Fonction *PGCD*(a,b)

a : entier	Premier argument de la fonction
b : entier	Deuxième argument de la fonction

1 *résultat = a*b÷PPCM(a,b)*

Fig. 1. Algorithme de réduction de deux fractions au même dénominateur : Copie de la solution proposée dans le bulletin vert APMEP n° 489

« Il n'y a pas un algorithme pour la fabrication d'algorithmes. Il y a des façons de faire » écrit J. Arsac, Il ne propose donc pas de méthode systématique, et pas de langage d'expression d'algorithme. On peut noter cependant qu'il utilise dans ses ouvrages un pseudo-code proche des langages de programmation impératifs de type PASCAL ou LSE. De façon plus générale, à cette période, la plupart des algorithmes étaient encore représentés graphiquement sous forme d'organigrammes. Les chercheurs reconnaissent aux organigrammes un pouvoir expressif certain, mais notent qu'ils ne sont d'aucune aide pour construire l'algorithme ainsi représenté, ils sont aussi un peu plus éloignés des langages de programmation, ce qui rend la phase de codage plus difficile.

Mais dans leurs différents articles de cette période, J. Arsac et C. Pair nous rappellent les problèmes rencontrés dans l'industrie du logiciel, attribués au manque de méthode pour la construction des applications, et l'arrivée de principes de génie logiciel pour assurer la construction de programmes plus sûrs dans des délais mieux maîtrisés. Il faut donc prendre de bonnes habitudes dès le début et apprendre à programmer avec méthode. Pour de futurs professionnels, cela ne fait aucun doute, pour la

formation générale, J. Rogalski (6) analyse les difficultés conceptuelles sous-jacentes à l'acquisition de méthodes chez les débutants.

3.3 Codage, vérification syntaxique et exécution

Plusieurs observations et propositions concernent la phase de codage, c'est à dire de traduction d'un algorithme exprimé avec un schéma ou un pseudo-code dans un langage de programmation. Certains auteurs du colloque (8), (16) voudraient l'éviter pour les premiers pas et proposent d'interpréter directement un langage algorithmique. Dans la logique de leur démarche, [Ducrin, 1984] proposent des schémas de traduction systématique des algorithmes MEDEE dans les langages utilisés à l'époque en France pour l'initiation à la programmation, c'est à dire BASIC, PASCAL, LSE.

En phase d'exécution, M. Romainville (15) montre comment des erreurs de logique interne se révèlent à ce niveau : variables non initialisées, nécessité de débogage, de test du comportement du programme.

Globalement cette phase d'exécution et de test est peu abordée, alors que c'est seulement avec l'exécution du programme que vont apparaître des imperfections au niveau de l'interface (quelle donnée faut-il entrer si on n'a pas pensé à faire afficher un message l'indiquant), des erreurs de logique dans le programme (pas de résultat affiché ou pas le résultat attendu). Enfin, les communications n'abordent pas non plus la lecture, la modification et l'exécution de code existant.

3.4 Quels prérequis et quelles représentations chez les élèves ?

Une quatrième préoccupation apparaît dans plusieurs communications et elle semble particulièrement pertinente dans le contexte d'hétérogénéité forte des classes actuelles de collège et de lycée. Elle constitue le fil conducteur du papier de C. Pair intitulé « l'apprentissage de la programmation » (7). L'introduction pose le problème : « La discipline à enseigner n'est pas alors le seul point de départ ; le professeur doit aussi être conscient du cadre préexistant chez l'élève. L'apprentissage a pour but de faire évoluer ce cadre pour le rapprocher plus ou moins de celui de l'expert, à travers des stades intermédiaires : la représentation que l'élève a de l'activité qu'il exerce va être confrontée à des expériences qui pourront la confirmer, l'enrichir, la remettre en question : il ne s'agit que du processus d'assimilation-accommodation de Piaget. » La suite du papier propose et illustre une décomposition de l'activité de construction de programme en stages successifs qui pourraient représenter autant d'états mentaux correspondants : Faire avec une machine, faire-faire, décrire un ensemble de calculs, décrire le programme, décrire de manière structurée, introduire les variables informatiques, etc.

B. Hauglustaine-Charlier (14) propose des animations vidéo comme support au cours donné par C. Duchâteau à de futurs enseignants. Il s'agit de concrétiser les métaphores utilisées dans le cours et de provoquer la construction d'images mentales chez les étudiants. Dans un autre ouvrage [Duchâteau, 90], cet auteur indique qu'il commence toujours ses cours en demandant à ses étudiants de dire ce que les mots « traiter des informations », qui interviennent dans la définition de l'informatique,

évoquent pour eux, avec l'objectif de relier les concepts nouveaux à ceux qu'ils ont déjà construits lors d'expériences antérieures. Un autre conseil pour amener son public au caractère formel des traitements informatiques est de travailler sur un continuum entre le facilement formalisable, par exemple calculer le prix à payer à la caisse du supermarché et le très difficilement formalisable, par exemple résumer un texte.

M. Romainville dans (15) parle aussi du besoin d'outils d'aide à la conceptualisation de certains processus sous-jacents à la programmation et suggère par exemple de fournir un modèle concret du fonctionnement d'une machine.

4 Plus récemment dans les revues internationales

Si des conférences internationales existent depuis 1970, les revues consacrées à l'enseignement de l'informatique apparaissent plutôt à la fin des années 1980, voire 1990. Pour les débutants en programmation, elles analysent essentiellement des cours d'un semestre d'introduction à la programmation en première année d'université, notamment aux USA. Les deux principales sociétés savantes américaines, ACM[7] et IEEE[8], ont constitué des groupes de travail sur l'enseignement secondaire, animé des conférences et proposé plusieurs versions d'un programme en informatique pour l'enseignement obligatoire (K12 curriculum). La revue ACM Transactions on Computing Education[9]créée en 2001 affiche comme objectif la publication de recherches sur l'enseignement de l'informatique à tous les niveaux de formation. Cependant, peu de communications traitent de l'enseignement scolaire, nous analysons ci-après des travaux traitant de formation des maîtres, d'élèves du niveau scolaire ou faisant une synthèse des difficultés des débutants avec l'objectif d'en transposer les conclusions vers l'enseignement secondaire.

4.1 Langage de programmation visuel ou textuel

La question de l'apprentissage des concepts de base de la programmation avec Scratch a été largement documentée. Par exemple [Armoni et al., 2015] publient une étude sur l'assimilation par des élèves de collège (middle school) d'un ensemble de concepts de programmation : initialisation de variables, boucles simples, boucles avec condition d'arrêt, envoi de messages, variables, parallélisme. De plus, l'utilisation de Scratch favoriserait la créativité, permettant ainsi d'atteindre l'un des objectifs de l'initiation à l'informatique chez les collégiens. On pourra consulter aussi [Kalelioglu et al., 2014] et les références du site de Scratch au MIT [Scratch].

Au delà de l'analyse du passage d'un langage visuel, Scratch pour les élèves de collège en France, au langage textuel Python proposé pour coder et exécuter leurs algorithmes à partir du lycée, d'autres modalités d'initiation à la construction de programmes méritent évidemment attention. C'est le cas du pilotage de robots ou autres

[7] ACM : Association for Computing Machinery
[8] IEEE Computer Society
[9] ACM Transactions on Computing Education : https://toce.acm.org/

objets tangibles et de l'utilisation de simulations numériques. Les robots sont proposés comme support possible pour atteindre les objectifs fixés aux cycles 2 et 3 de l'école primaire, le lecteur intéressé trouvera des références à ce sujet dans la bibliographie sélective commentée publiée par [Baron et al., 2016]. Une étude comparative, certes encore limitée, entre utilisation d'objet tangible ou de simulation numérique est proposée par [Fessard et al., 2019] et peut ouvrir la voie.

Enfin des modalités variées qui permettent d'aborder des sujets ludiques ou réels supposés plus stimulants ne doivent pas masquer la nécessaire unité des approches d'initiation à la programmation.

4.2 Contenu d'une formation pédagogique des enseignants

Les documents traitant de la méthodologie de l'enseignement de l'informatique sont assez rares, l'ouvrage « enseigner l'informatique » [Hartmann et al., 2012] traduit de l'allemand et rédigé par des enseignants suisses aborde de façon très pratique nombre de questions dans lesquelles les professeurs reconnaissent leur quotidien. Dans un article récent, [Yadav & Bergès, 2019] constatent qu'il y a partout dans le monde une demande très forte d'enseignants en informatique, des dizaines de milliers de professeurs à former, et font la proposition d'un test pour mesurer les connaissances pédagogiques nécessaires pour enseigner l'informatique. L'instrument se compose d'une liste de segments de travaux d'élèves (Scratch et Python) accompagnés de questions du genre « quelle explication donneriez-vous à ce stade du travail », « comment aideriez-vous cet élève ? ». Egalement à propos de pédagogie pour enseigner l'informatique au niveau scolaire, l'Académie des Sciences britannique (The Royal Society) a commandé une revue de littérature assez complète afin de synthétiser ce qui est connu à propos des pédagogies possibles pour enseigner l'informatique au niveau scolaire [Waite, 2017].

5 Actualité des questions et perspectives

Les difficultés des élèves dans leur activité de construction de programmes. Dans le programme de Sciences Numériques et Technologie pour la classe de seconde, la programmation apparaît comme notion transversale. On peut lire comme capacités attendues « Ecrire, exécuter et mettre au point un programme. » Il est rappelé ensuite que « Au collège (cycle 4) les élèves ont découvert et pratiqué les éléments fondamentaux d'algorithmique et de programmation. Le programme de seconde de mathématiques approfondit l'apprentissage de la programmation. » C'est donc au collège et au cours de mathématiques en seconde qu'il convient d'observer les difficultés des élèves dans leur activité de construction de programmes. Les compétences demandées en fin de collège sont illustrées par les sujets proposés à l'examen du Brevet des Collèges, on pourra consulter à ce propos l'étude réalisée par la commission inter-IREM informatique en octobre 2018 [Alayrangues et al., 2018].

La question de la construction d'un algorithme pour résoudre un problème donné est encore très ouverte, les images mentales que se forgent les élèves à propos des

programmes et de leur exécution sont importantes lors de leurs premiers contacts avec la programmation, elles devraient être mieux étudiées et les maîtres formés à en susciter. Elle pourrait être nourrie des observations faites par les professeurs de mathématiques et leurs collègues informaticiens à partir de certaines séquences du concours Castor[10] et déboucher sur une banque d'exercices ayant cet objectif, selon le niveau des élèves.

La place des méthodes est à redéfinir pour l'initiation. Pour les élèves qui choisissent le cours d'informatique en classe de première (NSI), le génie logiciel et ses phases de spécification, conception, architecture, test, maintenance doivent y avoir toute leur place tout au long des cours et des travaux pratiques.

La question des langages d'expression et du codage est loin d'être épuisée, elle peut être reprise à partir du passage de Scratch à Python, mais aussi plus directement et plus généralement en développant des outils pour évaluer les difficultés des élèves relativement aux concepts clé de la programmation.

La question d'habituer les élèves à lire du code existant dans le contexte du travail collaboratif ou de l'utilisation de bibliothèques de logiciels devrait aussi faire l'objet d'études et de recommandations. Commencer à faire programmer en modifiant du code existant est une approche souvent utilisée, cela suppose de savoir quand et comment « larguer les amarres » et proposer des exercices où tout est à construire, notamment le choix des données et de leur représentation

Enfin la question des prérequis et des représentations cognitives préexistant et évoluant chez les élèves ne semble pas avoir fait l'objet de beaucoup d'études et reste un champ important à défricher, en liaison avec les images mentales évoquées ci-dessus.

La formation des enseignants est évidemment un sujet transverse aux précédents et préalable à leur développement.

Pour l'enseignement primaire, il s'agit de sensibiliser à la pensée informatique en proposant des séquences réutilisables en classe. P. Tchounikine a mis en ligne un exemple possible de formation avec Scratch comme support [Tchounikine, 2017].

Au niveau du secondaire, plusieurs pays dont la France ont ouvert un grand chantier de formation continue dans des conditions difficiles, mais avec une implication remarquable des chercheurs, enseignants-chercheurs et professeurs du second degré et autres personnels de l'Education nationale, [Dowek et al., 2013] pour la spécialité ISN[11], puis [MOOC SNT, 2019], [DUI, 2019].

Pour les observations et recherches à entreprendre, [Hubwieser et al, 2015] ont coordonné deux numéros spéciaux rassemblant des expériences de développement de cours d'informatique au niveau secondaire dans différents pays afin d'offrir un panorama des initiatives récentes ou en cours. Dans leur introduction, ils abordent les questions de recherches ouvertes avec une impressionnante liste de 13 thèmes subdivisés chacun en plusieurs items, par exemple la qualification nécessaire pour les en-

[10] Castor Informatique France, http://castor-informatique.fr/

[11] ISN : Informatique et Sciences du Numérique, spécialité en terminales S et S-SI jusqu'en 2019

seignants et le cas du « tout formation continue », la motivation de l'élève et du professeur, les compétences qui doivent être mesurées et comment, la définition et l'évolution des objectifs et contenus de cours, le niveau de détail auquel il faut définir les contenus, la place du cours d'informatique dans les emplois du temps, etc. Dans cette longue liste, je retiens notamment le besoin d'instruments de mesure à long terme des compétences acquises aux divers niveaux de scolarité au travers d'activités individuelles mais aussi collectives.

Il faudrait aussi dépasser l'assimilation « didactique de l'informatique = didactique algorithmique-programmation » et mettre en évidence les difficultés d'enseignement posées par d'autres parties des programmes (représentation et gestion des données, internet et Web, réseaux, sécurité, etc.) ou celles plus générales liées à l'évaluation des compétences des élèves.

Enfin nous allons disposer de beaucoup de données d'apprentissage (modulo autorisations et anonymisation éventuelle). Ces données vont permettre de répondre à certaines des questions posées précédemment à propos des difficultés rencontrées, de leur contexte d'observation et des remédiations efficaces. Il est possible de tracer les essais/erreurs des élèves, d'analyser ces traces et de renvoyer des diagnostics et explications personnalisés. Ces nouvelles modalités de recherche n'ont pas encore été beaucoup utilisées pour documenter l'enseignement de l'informatique au niveau secondaire. Pour les débuts dans le supérieur, de telles études existent. Elles ont par exemple permis d'analyser les erreurs des programmeurs novices, y compris le temps passé à les corriger, de les catégoriser et de proposer la notion de « sévérité » d'une erreur [McCall & Kölling, 2019] comme produit de la fréquence et de la difficulté.

En conclusion, les questions évoquées pour les débutants en 1988 restent d'actualité, même si elles doivent être revisitées dans le contexte actuel. Beaucoup d'autres doivent être abordées, notamment pour couvrir les différents aspects des programmes. Comme souligné en 1988, ce travail nécessite une étroite collaboration entre chercheurs et praticiens, collaboration qui doit être organisée dans des programmes et cadres spécifiques qu'il est urgent de mettre en place.

Références

1. Alayrangues S. et al., Une analyse des exercices d'algorithmique et de programmation du brevet 2017, prépublication, https://hal.archives-ouvertes.fr/hal-02077738, (2019)
2. Armoni M., Meerbaum-Salant O., Ben-Ari M., From Scratch to « Real » Programming, *ACM Trans. Comput. Educ.* 14, 4, Article 25, 15 p., (2015)
3. Arsac J., La Science informatique, Dunod, Paris, (1970)
4. Arsac J., Premières leçons de programmation, Cedic, Paris, (1980)
5. Arsac J., Des pédagogies pour l'option informatique des lycées, in Pratiques et Savoir-faire des élèves de l'option informatique des lycées, DLC, ministère Education Nationale, Paris, (1987)
6. Baron G.-L., Baudé J., Cornu P. eds., Actes du premier colloque francophone de didactique de l'informatique, publié par Enseignement public et Informatique (EPI), http://www.epi.asso.fr/association/dossiers/d07som.htm, (1988)

7. Baron G.-L., Drot-Delange B., Touloupaki S., L'éducation à l'informatique à l'école primaire : une bibliographie sélective commentée. Adjectif.net, [En ligne] http://www.adjectif.net/spip/spip.php?article382, (2016)
8. Dijkstra E. W., A discipline of programming, Prentice Hall, New York, (1976)
9. Dowek, G., Archambault, J. P., Cimelli, C., Wack B., Baccelli, E., Cohen, A., Eisenbeis, C., Viéville T., Informatique et sciences du numérique, Eyrolles, 342 p., (2013)
10. Duchâteau C., Images pour programmer, Apprendre les concepts de base. De Boeck-Wesmael, Bruxelles, (1990)
11. Ducrin A. (nom collectif), Programmation, Tomes 1 et 2, Dunod, Paris, (1984)
12. DUI : Diplôme Inter Universitaire Enseigner l'Informatique au lycée, https://sourcesup.renater.fr/www/diu-eil/, consulté déc. 2019, (2019)
13. Fessard G., Wang P., Renna I., Objet Tangible ou Simulation Numérique: Deux Situations Équivalentes pour l'Apprentissage de la Programmation? In Environnements Informatiques pour l'Apprentissage Humain, Juin 2019, Paris, France, hal.archives-ouvertes.fr/hal-02156174/file/EIAH_2019.pdf, (2019)
14. Gries D., The science of programming. Springer Verlag. Berlin, (1981)
15. Hartmann W., Näf, M., Reichert, R., Enseigner l'informatique, (édition originale en allemand parue en 2006), Springer, Berlin, (2012)
16. Hubwieser P.et al, How to implement rigorous computer science education in k-12 schools ? Some answers and many questions, *ACM Trans. Comput. Educ.* 15, 2, Article 5, 12 p., (2015)
17. Kalelioğlu F., Gülbahar Y., The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective, *Informatics in Education*, Vol. 13, No. 1, 33–50, Vilnius University pub., (2014)
18. Langer B., MEDEE : Une méthode pour construire des algorithmes, Le bulletin Vert, APMEP 489, pp. 403-412, consulté déc. 2019 à l'URL http://numerisation.univ-irem.fr/AAA/AAA10055/AAA10055.pdf, (2010)
19. McCall D., Kölling M., A new look at novice programmer errors, *ACM Trans. Comput. Educ.* 19, 4, Article 38, 30 p., (2019)
20. MOOC SNT, S'initier à l'enseignement en Sciences Numériques et Technologie, https://www.fun-mooc.fr/courses/course-v1:inria+41018+session01/about, (2019)
21. Nguyen, C. T., Etude didactique de l'introduction d'éléments d'algorithmique et de programmation dans l'enseignement mathématique secondaire à l'aide de la calculatrice. Thèse de doctorat. Université Joseph-Fourier-Grenoble I. http://tel.archives-ouvertes.fr/tel-00011500/, (2005)
22. Pair C., Je ne sais (toujours) pas enseigner la programmation, *Informatiques* 2, 5-14 (1988)
23. Samurçay R. Signification et fonctionnement du concept de variable informatique chez les élèves débutants, *Educational Studies in Mathematics*, 16, pp. 143-161, (1985)
24. Scratch, https://scratch.mit.edu/research, consulté déc. 2019
25. Tchounikine P., Initier les élèves à la pensée informatique et à la programmation avec Scratch, http://lig-membres.imag.fr/tchounikine/PenseeInformatiqueEcole.html, (2017)
26. Waite J., Pedagogy in teaching Computer Science in schools : A Literature Review, added to The Royal Society Computing Education Project Report, https://royalsociety.org/~/media/policy/projects/computing-education/literature-review-pedagogy-in-teaching.pdf, (2017)
27. Wirth N., Algorithms + Data Structures = Programs, Prentice Hall, New York, (1976)
28. Yadav A., Berges M., Computer Science Pedagogical Content Knowledge: Characterizing Teacher Performance. *ACM Trans. Comput. Educ.* 19, 3, Article 29, 24 p., (2019)