ÉTUDE DE L'ÉVOLUTION DES MÉTHODES D'APPRENTISSAGE ET DE PROGRAMMATION

Christophe CAIGNAERT

Je vais ici essayer de faire le point sur les diverses méthodes d'apprentissage de la programmation et les divers types de pratiques correspondantes. Il s'agit de tenter de les répertorier et de les classer selon un certain nombre de caractéristiques à déterminer ; en cela, il s'agit aussi de répertorier et classer les outils qu'elles utilisent, encore une fois selon un certain nombre de critères à déterminer.

Je vais commencer par exclure de cette étude l'utilisation du LOGO au sens Papertien/Piagétien du terme [1]. Non parce que cette pratique serait inintéressante ou inefficace, mais parce qu'elle déborde de mon propos. C'est d'une théorie globale de l'apprentissage dont il s'agit, alors qu'on ne parle ici que d'apprentissage de la programmation. Ces deux apprentissages sont liés mais ne peuvent se confondre. Il est clair que tout type d'apprentissage particulier rentre, consciemment ou non, dans une théorie globale de l'apprentissage.

PHASE ZÉRO : AVANT LE DÉBUT

Les algorithmes ont "toujours" existé, au moins en mathématiques depuis, par exemple, Euclide, Erastosthène etc. L'algorithmique (la programmation ?) n'a pourtant jamais été véritablement enseignée avant la naissance de l'informatique : il ne servait le plus souvent pas à grand chose de créer de nouveaux algorithmes qu'on ne pouvait appliquer autrement qu'à la main. Cependant les structures actuelles utilisées en programmation étaient déjà présentes : il y a du "tant que" dans l'algorithme d'Euclide pour trouver le PGCD de deux nombres [2], de

(1) Voir "Le jaillissement de l'esprit" de Seymour Papert, Flammarion et aussi "LOGO, des ailes pour l'esprit" de Horacio C. Reggini, Cédic.

(2) Que cet algorithme soit du à Euclide ou non ne change rien, il est antérieur à l'apparition du premier ordinateur.

même, on peut considérer qu'il y a de la récursivité dans l'algorithme de l'addition habituel !

Puisqu'il n'y avait aucun intérêt à enseigner l'algorithmique, on ne le faisait pas, l'algorithmique est ainsi, en tant que discipline, restée longtemps "naïve", non théorisée, non "fondée".

PHASE UN : AU DÉBUT, L'EMPIRISME

Ecoutons J. Arsac : "On enseignait un langage de programmation, et on aidait l'apprenti à organiser ses idées par l'organigramme. (...). On enseigne aux gens "comment dire", pour le reste, ils sont autodidactes, ou l'apprennent ailleurs" [3].

Ça a été la première méthode, on enseignait un "langage de programmation". J. Arsac a sans doute tort de parler au passé, cette méthode s'utilise toujours. Dans combien d'écoles et de collèges voit-on un cours de Basic, et même de Logo...

Cette méthode n'a pas que des inconvénients, c'est la seule que je connaisse qui donne lieu à évaluation simple, facile à mettre en oeuvre et objective !... Il et facile d'interroger sur la syntaxe du SCREEN ou la façon de changer les couleurs de l'écran du TO7... J'ai été témoin d'une "composition" de LOGO qui nécessitait d'apprendre par coeur les seize codes de couleur du MO5. Pour beaucoup, il s'agit là d'un avantage qu'aucune autre méthode ne pourra jamais combler.

Il y a au moins un autre avantage : c'est la seule méthode utilisable quand on précède soi-même ses élèves d'une semaine dans l'apprentissage... tant qu'il restera de très nombreux maîtres peu ou mal formés, elle subsistera. L'enseignement d'une discipline quelle qu'elle soit nécessite, évidemment un certain recul, une certaine culture. Il faut bien dire que ce n'est pas toujours le cas.

Ceci dit, cette méthode reste également plus ou moins sous-jacente à tous les ouvrages du type : "La programmation, c'est pas sorcier" et "Le Basic facile" [4], j'en passe et des meilleurs. Ces livres sont d'ailleurs, bien que très chers, de grands succès de librairie. Si ce n'était pas sorcier, aurait-on besoin d'eux ?

---

(3) Extrait de la préface de "Programmatique" par C. et P. Richard, Belin.

(4) Quoi de plus simple que 10 PRINT "BONJOUR" (ENTRÉE) RUN (ENTRÉE). Vous voyez que vous savez faire un programme !

Enfin, signalons que cette pratique empirique est en fait, en partie (bien sûr, en partie seulement), à l'origine du Basic ; l'un des concepts de base de ce langage pourrait s'écrire :

"Pour simplifier la programmation, simplifions le langage !"

Le "B" de "BASIC" signifie en effet "Beginner", c'est-à-dire débutant. On peut ouvrir le débat, mais ce n'est pas le lieu ici, pour savoir si le Basic est simple ou simpliste.

PHASE DEUX : RÉACTION Ð L'EMPIRISME, L'ALGORITHME

Rendons à César ce qui lui appartient, l'algorithmique classique a été la première à se poser de vrais problèmes pédagogiques. Constatant l'impuissance de la méthode empirique, des gens comme Edsger Dijskstra, Niklaus Wirth (qui créera Pascal entre autres), Henry Ledgard et en France Jacques Arsac ont amené l'idée que la programmation était une résolution de problème et qu'il fallait s'y prendre avec METHODE [5].

Descartes était appelé plus d'une fois à la rescousse, comme à chaque fois qu'un problème peut être divisé en plusieurs problèmes plus petits, ce qui donne le premier concept de programmation descendante. Avant d'écrire un programme, on écrit un algorithme ; en fait, d'ailleurs, on n'écrit même pas un programme, on programme un algorithme... C'est plus qu'une nuance.

L'idée était que puisqu'on programmait un algorithme, les langages devaient être conçus pour programmer des algorithmes. On a ainsi créé des langages destinés à programmer des algorithmes : Algol (ALGOrithmic Language) dans les années soixante, suivi de Pascal dans les années soixante-dix (et de beaucoup d'autres) ; s'est d'autre part développé le refus motivé de nombreux enseignants, universitaires et autres, d'utiliser le Basic en initiation.

Ce courant a donc fortement marqué l'évolution des langages [6], mais les langages ne sont plus alors que des utilitaires dont certains enseignants aimeraient pouvoir se passer. A quoi sert-il de coder un langage alors que le problème essentiel est d'écrire l'algorithme ? Pour eux, la discipline, c'est l'algorithmique, pas la programmation.

---

(5) Notons que parler de "méthode empirique" n'a aucun sens, l'empirisme étant justement l'absence de méthode...

(6) et est d'ailleurs fortement marqué par l'évolution des langages...

Ceci dit, l'apport principal de l'algorithmique et la généralisation de la notion de structure à tous les niveaux : structures de programme (procédures), structures de contrôle, structures de données. Généralisation associée à la prise en compte de l'importance fondamentale de ces structures et la recherche de concepts pour manier ces structures (invariants, schémas de preuve de programme, transformation automatique de programmes...).

Cela a conduit tout naturellement à la programmation structurée, souvent naïvement confondue avec la programmation "modulaire" ou descendante (Descartes !...). On a donc développé des langages et, à l'autre bout de la chaîne, des méthodes pour écrire, décrire et manipuler des algorithmes. On peut citer les arbres programmatiques et la méthode Aladin par exemple.

La méthode algorithmique fut, on l'a déjà dit, la première à s'occuper de "pédagogie", elle est très marquée pédagogiquement : Pascal possède par exemple des contraintes très fortes, on est parfois obligé de le "tromper" quand on veut par exemple accéder à une case mémoire particulière de l'ordinateur ; Pascal est alors "pédagogique" au point d'en devenir ennuyeux !

Le problème, car problème il y a, est que cette méthode s'est développée en dehors de toute "théorie" de l'apprentissage. Pour ses concepteurs, il y a une vérité de programmation [7]. On détermine -théoriquement- ce que c'est que programmer, donc ce qu'il faut apprendre quand on apprend à programmer ; on a alors des contenus d'enseignement. Il n'y a plus qu'à établir une progression de ces contenus qu'on prévoit plus ou moins rapide selon le public.

La programmation doit alors se faire comme on a déterminé qu'elle devait se faire ; point à la ligne. Le discours est fermé, du type : "L'important, c'est l'algorithmique. Les notions de base de l'algorithmique sont : 1) la structure séquentielle 2) etc. Il suffit donc d'"expliquer" clairement la chose pour que les gens comprennent ; s'ils ne comprennent pas, il suffit de réexpliquer encore plus clairement ou de revoir la progression [8].

(7) On ne peut pas d'ailleurs ne pas penser aux mathématiques modernes qui relevaient pour une grande part du même principe.

(8) D'où un dogmatisme insupportable, il y a un "noyau dur" jamais remis en cause. Ainsi, on considère que l'échec est toujours du à la progression mal adaptée ou au public qui, de toutes façons, ne comprendra jamais.

L'expérience a tendance à montrer que cela peut fonctionner avec un public adulte très volontaire et motivé au départ (il se pose des questions par rapport à ce qu'il est en train de faire et est capable à priori de manier les concepts utilisés) et/ou scientifique (habitué au type d'objet manipulé). Si il y a cinq ans, le public répondait presque toujours à ce critère, ce n'est plus le cas aujourd'hui, loin s'en faut (c'est même l'inverse). Il y a un très net changement de population concernée, j'ai personnellement à initier à la programmation des enfants d'une dizaine d'années ou des instituteurs, la plupart sans passé scientifique universitaire. Le champ social de l'apprentissage de la programmation s'est ainsi radicalement transformé.

On se retrouve donc dans la situation dans laquelle on est à chaque fois qu'on a déterminé à priori, sans tenir compte du public, un type d'apprentissage : le plus souvent, ça ne marche pas [9]. Ce type d'apprentissage est ici uniquement déterminé par le contenu, il devient donc absolu, souvent dogmatique et donc sans doute dangereux. Combien de fois dans le Nord n'avons nous pas été confrontés à l'obligation d'utiliser exclusivement des arbres programmatiques ! Tout autre moyen d'expression, toute mise en cause étant bannis a priori. J'ai toujours considéré cette position comme ridicule car fermée, mais qu'est-ce qu'un petit professeur de mathématiques pouvait apporter au discours universitaire des spécialistes de la chose ? Le danger "dogmatique" est donc d'autant plus grand que la méthode est validée universitairement, car d'origine universitaire, et que donc le mythe de la "vérité scientifique" joue à plein [10]. En fait, il manque une réflexion didactique : la prise en compte du public tel qu'il est face aux contenus d'enseignement.

Avant de passer à la suite, je voudrais signaler un autre point, qui me pèse : beaucoup de langages se sont développés, c'est vrai, sous l'influence de l'algorithmique, mais pas tous ! Je cite J. Arsac dans l'avertissement des "premières leçons de programmation" (Cédic) : "la façon de concevoir un programme est à peu près indépendante du langage (ceci a des limites, nous ne conseillons pas ce livre a ceux qui

(9) Il y a d'ailleurs une évolution nette dans certains propos, J. Arsac prend conscience de cet échec : "S. Michaelson rapproche l'apprentissage de cette discipline de celui des anciennes formes de l'artisanat : l'apprenti travaillait auprès du maître, le regardant faire, et finissait par s'approprier sa propre maîtrise". Cette pratique est clairement antipodique du cours magistral d'algorithmique. De plus, un maître ne peut avoir que quelques disciples : (Extrait du document noté en 3).

(10) Je ne conteste pas ici l'apport scientifique de l'algorithmique mais l'utilisation qui en est faite en apprentissage de la programmation. Encore une fois, c'est plus qu'une nuance.

voudraient utiliser LISP, voir même APL)". J'ajouterais LOGO et Prolog au minimum. On peut alors se demander si les gens qui font du LISP programment réellement !... Parfois, on a l'impression qu'il faudrait supprimer les langages pour lesquels la méthode ne s'applique pas. La confrontation des langages ne peut, à mon avis qu'être bénéfique à tous.

PHASE TROIS : REFUS DU DOGME ET DIDACTIQUE

L'important ici est la confrontation du public et des notions qu'il doit acquérir [11]. On y retrouve principalement comme animateur des gens qui ont constaté qu'avec leur public, algorithmique pure et dure, ça ne marchait pas. Ils ne refusent pas l'algorithmique classique, ils s'en servent comme acquis culturel, comme point de départ. Le public visé se trouve pour des raisons déjà données hors de l'enseignement scientifique traditionnel.

En premier lieu, il convient de tenir compte des problèmes posés par l'enseignement de la programmation et/ou de l'algorithmique. On peut par exemple déterminer que les notions à faire acquérir sont :

- Les notions de procédures
  de structures de contrôle
  de variable.

Le travail consiste à chercher des activités qui pourront amener les gens à définir des procédures, créer des variables ou utiliser des structures de contrôle.

La différence avec l'algorithmique classique vient de qu'on ne cherchera pas à étudier toutes les structures de contrôle ou tous les types de variable mais à ce que les élèves utilisent effectivement, c'est-à-dire comprennent, celles qu'on étudie. La question importante est celle du "comment", par exemple, "comment" choisir les variables dans une situation donnée, "comment" faire...

En effet, le but n'est pas de faire des programmeurs (ce qui serait ridicule à une telle échelle, on n'a aucun besoin de millions de programmeurs), il est de développer la "pensée logique", la capacité d'abstraction, par un apprentissage de la programmation. Objectif d'ailleurs beaucoup plus ambitieux puisque non notionnel mais

(11) Attention, il ne s'agit pas de choses du type : "ce sont des littéraires/enfants/... alors je vais moins vite..."

structurel. Beaucoup plus ambitieux et donc beaucoup plus difficile à atteindre et à évaluer.

Je vois encore deux différences importantes avec l'algorithmique classique, différences qui sont d'ailleurs des conséquences de cet apprentissage -par- et non pas -pour- la programmation :

1 Il est important d'étudier avec soin les productions des élèves, afin de déterminer comment "ça marche dans leur tête" et de repartir de ces observations [12]. La façon de procéder des élèves est alors au moins aussi importante que les programmes produits.

2 Il n'y a plus alors aucun intérêt à s'attacher à un langage particulier et à un mode d'expression particulier d'algorithmes.

Beaucoup de langages sont utilisables, on peut citer par exemple :

- Le langage naturel, car ce qui se conçoit bien, s'énonce clairement et les mots pour le dire arrivent aisément. Pour des situations très complexes où il serait impossible d'aller jusqu'au détail, et parce que le langage naturel supporte ellipses et imprécisions, ce qui est bien pratique parfois.
- Le Basic, pour sa facilité de mise en oeuvre, son graphisme et ses variables.
- Le LOGO, pour ses procédures et son extensionnalité.
- Un tableur comme COLORCALC, pour la notion de variable, la différence entre son nom (en fait ici sa position), sa valeur (ce qui est affiché) et la façon dont elle est calculée (sa formule). Egalement pour un autre regard sur la séquentialité. Et pour la qualité des documents produits sans aucun PRINT USING, aucune virgule ni aucun point-virgule!...
- Un gestionnaire de fichiers pour, par exemple structurer une suite de conditions logiques.
- Des macro-procédures, c'est-à-dire un "micro-langage" adapté à une situation complexe particulière : feux de circulation, gare de triage.... Situation qui serait impraticable dans un langage de programmation habituel à ce niveau.
- Une calculatrice ordinaire par exemple dans le développement de stratégie pour calculer des expressions complexes.

(12) On lira avec intérêt l'article de Robert Neyret dans la revue Grand N, de l'Irem de Grenoble, numéro 37.

Il ne s'agit pas de faire l'inventaire de ce que l'on peut faire avec un certain produit (Basic, LOGO, tableur...) mais de bien voir que chacun de ces produits peut servir à atteindre certains objectifs fixés dans une initiation à la programmation. Il ne s'agit pas non plus que chaque élève manipule nécessairement chacun de ces outils, ce qui serait probablement stupide. En fonction des objectifs que l'enseignant s'est assigné, il va choisir certains des supports décrits, ou d'autres. Il est ainsi relativement classique aujourd'hui d'utiliser LOGO pour travailler sur la notion de procédure et Basic pour les variables.

Mais l'analyse du fonctionnement d'un ascenseur, par exemple, ne peut se faire que si on a pris conscience qu'il y a des variables (appel depuis chaque palier, position et état de l'ascenseur, appel depuis la cabine) et que le mouvement de l'ascenseur va être déterminée à un instant donné par l'état de ces variables. Une analyse en langage courant peut ainsi, par exemple, parfaitement servir l'acquisition de la notion de variable.

BILAN PARTIEL, PARTIAL ET PROVISOIRE

L'évolution de l'apprentissage de la programmation est intéressante à observer car, parti de rien il y a trente ans, cet apprentissage concerne aujourd'hui tous les enfants de dix ans. Parti de rien, on est également parti sans méthode, devant les résultats catastrophiques obtenus, les scientifiques ont durement réagis créant une nouvelle discipline : l'algorithmique.

Les baisses spectaculaires et imprévisibles [13] des prix des matériels ont entrainé une généralisation de cet apprentissage dans une phase très récente, le changement de type de clientèle implique un changement d'objectifs donc de démarche. La méthode algorithmique, toujours utilisable avec la clientèle pour laquelle elle a été conçue n'est pas universellement transposable. Ainsi, la plupart du temps, on n'enseigne plus la programmation pour former des programmeurs, comme on n'enseigne pas les mathématiques pour former des mathématiciens. En un mot, non pas pour mais par la programmation.

(13) Il est amusant à ce sujet de relire de vieux numéros de Sciences et Vie imaginant l'an 2000 : On y trouve beaucoup de choses encore maintenant très futuristes, mais jamais d'ordinateurs domestiques... ni de machine à laver pilotée par microprocesseur.

C'est ce virage qu'il faut savoir négocier sous peine de retomber dans l'empirisme des débuts. Le devenir actuel des mathématiques moderne nous indique a priori combien peuvent coûter de telles erreurs.

Nous sommes avertis, cela suffira-t-il ?

Christophe CAIGNAERT
École Normale de Lille

BIBLIOGRAPHIE

- **J. Arsac** : La construction de programmes structurés, Dunod.
- **J. Arsac** : Premières leçons de programmation, Cédic-Nathan.
- **O. Arsac, C. Bourgeois & M. Gourtay** : Premier livre de programmation, Deuxième livre de programmation, Pour aller plus loin en programmation, Cédic-Nathan.
- **J. Biondi & G. Clavel** : Introduction à la programmation, Tome 1 & 2, Masson.
- **C. Caignaert** : Pour une didactique de la programmation 1, 2, 3 & 4, Vous vous changez, changez de langage, Ronéos Ecole Normale de Lille.
- **H.F. Ledgard** : Proverbes de programmation, Dunod.
- **H.F. Ledgard** : Ada - Une introduction, Masson.
- **R. Neyret** : Situation problème et programmation, Revue Grand N n°37, CRDP Grenoble.
- **S. Papert** : Le jaillissement de l'esprit, Flammarion.
- **C. & P. Richard** : Initiation à l'algorithmique, Belin.
- **C. & P. Richard** : Programmatique, Belin.
- **P.C. Scholl** : Algorithmique et représentation des données, Tomes 1, 2 & 3, Masson.
- **XXX** : Informatique pour tous , CNDP-MTN.