Une analyse critique de l'initiation a l'informatique : quels apprentissages et quels transferts ?

Marc Romainville

UNE ANALYSE CRITIQUE DE L'INITIATION A L'INFORMATIQUE : QUELS APPRENTISSAGES ET QUELS TRANSFERTS ?

M. Romainville

Facultés Universitaires Notre-Dame de la Paix
Département Education et Technologie
Rue de Bruxelles 61
5000 NAMUR

UNE ANALYSE CRITIQUE DE L'INITIATION A L'INFORMATIQUE : QUELS APPRENTISSAGES ET QUELS TRANSFERTS ?

M. Romainville

Facultés Universitaires Notre-Dame de la Paix
Département Education et Technologie
Rue de Bruxelles 61
5000 NAMUR

INTRODUCTION : une vague déferlante

Si l'on analyse les rapports établis ces dix dernières années entre l'enseignement et l'informatique, il est curieux de constater que si les discours sur l'E.A.A.O. se sont multipliés à une allure vertigineuse, sa pratique est restée relativement rare sans doute à cause du manque évident de matériel et de didacticiel de qualité.

A l'inverse, on parle peu de l'initiation à l'informatique ; or, cette activité s'est répandue à une vitesse impressionnante à tel point que son bien-fondé est en général reconnu par tous. En Belgique, par exemple, il existe désormais des cours à option d'initiation à l'informatique dans le secondaire ; le décret RISOPOULOS recommande la mise au point d'une alphabétisation informatique dès l'enseignement primaire. Les parents, soucieux de ne pas rater le "train du progrès" recommandent une formation à l'informatique. Enfin, les enseignants eux-mêmes semblent se rallier à cet avis : une enquête réalisée auprès des enseignants du fondamental de l'Etat montre, par exemple, que 82 % d'entre eux marquent leur accord pour la proposition suivante "Promouvoir l'informatique dans l'enseignement fondamental est indispensable."[1]

Et l'on pourrait ainsi multiplier les exemples ; l'essentiel est de montrer qu'il se réalise une certaine unanimité quant au bien-fondé de cette initiation générale.

Nous avons voulu rassembler dans cette communication quelques réflexions critiques concernant cette vague d'alphabétisation informatique. L'objectif est d'en montrer les leurres et les difficultés et de tenter d'aller plus loin dans la définition des contenus et des méthodes liés à cette initiation.

---

(1) M.E.N., Organisation des Etudes, Informatique et enseignement fondamental de l'Etat.

QUELS OBJECTIFS ?

Analysons de plus près ce phénomène en essayant de repérer les motifs invoqués par ces ardents défenseurs d'une initiation à l'informatique.

Dans la plupart des documents que nous avons récoltés[2], nous retrouvons le même type d'exposé des motifs : dans un premier temps, les auteurs insistent sur un objectif d'"éveil au phénomène socio-culturel et technologique que constitue l'informatique". Il s'agit donc essentiellement, dans le cadre des activités d'éveil, de montrer aux enfants l'importance que l'informatique a prise dans notre société, son rôle, ses effets positifs et négatifs, de les familiariser avec la machine, ses composants et sa structure.

Dès cette première étape, on pourrait se poser un certain nombre de questions :

1. Si les activités d'éveil ont pour objectif d'aider les enfants à appréhender un certain nombre de données ou de dimensions de leur environnement, il est de fait assez logique qu'actuellement la dimension informatique soit prise en compte. Mais pourquoi pas d'autres ? En effet, d'autres révolutions technologiques ont profondément influencé nos sociétés : le téléphone et l'audio-visuel de masse pour ne citer que deux exemples. Existe-t-il, dans les activités d'éveil, des séances d'alphabétisation téléphonique ou audio-visuelle ?
2. Plus sérieusement, quel sera le contenu de l'éveil technologique ? En effet, si le contenu de l'éveil socio-culturel peut aisément être délimité (analyse et observation d'une société informatisée), l'éveil technologique pose plus de problèmes :

- Jusqu'où aller ?

Il paraît impossible de vouloir réellement faire comprendre le détail du fonctionnement électronique d'un ordinateur à des jeunes enfants. Ce niveau d'explication n'est d'ailleurs probablement pas pertinent. Néanmoins se pose la question de savoir jusqu'à quel degré de précision doivent parvenir les métaphores fournies aux apprenants pour expliquer le fonctionnement général de la machine : si celles-ci restent trop générales, l'apprenant sera-t-il en mesure de comprendre certains phénomènes observés ? (Ex. : le fait qu'en mode direct LOGO, ce qui est affiché n'est pas connu de la tortue, ne peut être compris que si l'on sait que la gestion de l'affichage à l'écran se réalise à l'aide d'une mémoire spécifique indépendante). J. WEIZENBAUM, nous a habitué depuis bien longtemps à de pareilles critiques

(2) Cf. Liste des documents en annexe 1.

concernant les tentatives d'explication du fonctionnement du hardware :

> *"Savez-vous comment fonctionne un téléphone ou un réfrigérateur ? Non. Et pourtant vous vous en servez tout le temps. Demain, quand vous utiliserez le computer ou la machine à traitement de texte, vous ne saurez pas non plus comment ça marche. Il suffira d'appuyer sur le bouton."*(3)

- A quelle technologie initier ?

  Celle-ci ne sera-t-elle pas complètement dépassée une fois le cours terminé ?

Mais assez rapidement, les auteurs passent, dans un second temps, à un motif qu'ils jugent souvent plus important : au-delà de cette première alphabétisation, on découvre alors un ensemble d'objectifs plus ambitieux concernant le développement de la pensée logique - voire de la pensée tout court - à travers la programmation. Prenons un exemple tiré des conclusions du congrès de la C.N.A.P.

> *"La rigueur, résultant de la démarche algorithmique, fondement de l'informatique constitue un apport indispensable à la formation des élèves de l'an 2000."*(a)

L'ensemble des documents analysés mettent donc bien l'accent sur ce que l'on pourrait appeler la "programmation-prétexte". Car partout, il est bien rappelé que ce n'est pas l'apprentissage d'un langage qui est visé, mais bien le développement des capacités cognitives logiques supposées être la base de tout travail de programmation structurée. Ces "skills" de résolution de problèmes sont, entre autres, les capacités à poser un problème, à le décomposer en sous-problèmes, à planifier des actions, à anticiper des résultats (formuler des hypothèses), etc.

Un des relevés des plus complets des effets cognitifs attendus de l'activité de programmation a été réalisé par FEURZEIG (cité par PEA et KURLAND p.143).

1) la rigueur de la pensée et la précision de l'expression ;
2) la compréhension de concepts généraux tels que variable, fonction ;
3) une plus grande facilité dans l'utilisation d'heuristiques, c'est-à-dire d'approches générales de résolution de problèmes telles que la planification, la décomposition en sous-problèmes ;
4) la conception du "debugging" (recherche des erreurs) comme une étape constructive et planifiable ;

---

(3) Extrait d'une interview de J. WEIZENBAUM : L'ordinateur à l'école : une plaisanterie, Le Nouvel Observateur, n°2228, décembre 1983.

5) l'attitude générale de chercher une solution à un problème important en construisant petit à petit des solutions partielles ;
6) la réflexion et la prise de conscience des processus de résolution de problèmes ;
7) la prise de conscience du fait qu'il n'y a que rarement une seule "bonne" solution mais plus souvent différents chemins avec chacun leurs avantages et inconvénients.

Voilà donc bien le vrai motif et l'on se rend compte alors qu'il n'est pas spécifiquement lié à la technique informatique mais qu'il vise ces fameuses capacités cognitives de base. Il est aussi évident que la haute qualité de ces objectifs provoque l'unanimité et ... malheureusement clôture les documents comme si le seul énoncé des objectifs entraînait inexorablement leur réalisation.

L'objectif de la communication est de montrer que la réalisation de ces objectifs est loin d'être automatique et que l'optimisme sans limite concernant les bénéfices cognitifs de la programmation occulte une discussion de fond sur le bien-fondé de cette initiation et sur les conditions précises dans lesquelles des transferts seraient possibles.

Si ce travail ne se réalise pas, le risque est grand de voir l'initiation à l'informatique, justifiée par l'annonce d'objectifs de hauts niveaux, se réduire à un cours mi-historique (les premiers ordinateurs, etc.), mi-technique (qu'est-ce qu'une RAM, ROM, etc.)[(4)].

QUELQUES PROBLEMES ...

En simplifiant un peu, le motif fondamental est donc le suivant : l'enfant, en apprenant la programmation, apprendra et exercera des techniques de résolution de problèmes. On peut déceler deux niveaux de questions dans cette proposition : premièrement on suppose que l'enfant apprendra sans trop de difficultés la programmation, deuxièmement on suppose que, cela étant fait, il aura appris des techniques transférables à d'autres situations.

Un certain nombre de faits vont cependant relativiser très fortement ces deux affirmations.

(4) Un exemple de cette "réduction" nous est donné par le livre de F. GANGLOFF "L'enfant apprenti programmeur" (EYROLLES, 1986). Les titres des leçons, à eux seuls, sont significatifs : quatrième leçon : l'instruction LOCATE et son utilisation, quatorzième : l'utilisation des INPUT ; le point virgule, etc.

1. Les problèmes rencontrés dans l'apprentissage de la programmation

Un certain nombre d'études montrent que - contrairement au mythe de l' "enfant programmeur" - les débutants en programmation éprouvent un certain nombre de difficultés importantes. Celles-ci peuvent être regroupées en trois moments (DU BOULAY B. et O. SHEA T.)

a. La planification : les erreurs ont trait, dans ce cas, au découpement logique et préalable de la tâche. Ex. : sous-spécifier le problème : ne pas prévoir ce qui se passe quand telles conditions ne sont pas remplies.
b. Le codage : il s'agit des erreurs concernant le langage utilisé (sa syntaxe, sa logique interne, etc.). Ex. : erreur de logique interne au langage : lire un fichier sans l'avoir ouvert.
c. Le debugging .

En LOGO, par exemple, des enfants de 9 à 15 ans éprouvent des difficultés dans les domaines suivants : (DU BOULAY B. et O.SHEA T. J.)

1) manque de spécifications ;
2) confusion des rôles de la machine :
   - chargeur de LOGO ;
   - exécution ;
   - édition ;
   - utilisation d'un programme LOGO.
3) croyance que le système stocke automatiquement un certain nombre de données ;
4) distinction nom/valeur d'une variable, d'une procédure. Ex. : croire qu'un nom de procédure doit obligatoirement signifier ce qu'elle fait ;
5) distinction répétition/récursivité ;

   etc.

Remarquons au passage qu'il ne s'agit pas souvent d'erreurs de syntaxe ou de règles sémantiques.

Les auteurs notent également que les enfants ne décomposent pas assez leurs problèmes, construisent des procédures peu élégantes, n'utilisent pas les solutions apprises dans de nouvelles situations. Enfin ces problèmes semblent être d'une importance très variable d'un enfant à l'autre.

Bref, il n'est pas sûr qu'un certain nombre de concepts riches (variable - récursivité) soit réellement maîtrisé par la plupart des enfants.

Un bel exemple de représentation erronée et systématique d'un concept de base est analysé dans un article de KURLAND et PEA. Il concerne la manière dont se déroule une procédure récursive en LOGO.

> *"La plupart des enfants croient que le fait de placer le nom d'une procédure à l'intérieur d'elle-même provoque l'exécution d'une boucle à l'intérieur de la procédure, alors qu'en réalité, le contrôle passe à une copie de la procédure. Cette procédure est exécutée et, quand le processus est terminé, elle repasse la main à la procédure qui l'a appelée en dernier lieu.*
>
> *Les enfants adoptent des modèles mentaux de déroulement du programme qui marchent pour des cas simples, tels que des programmes constitués d'une seule procédure, mais qui se révèlent inadéquats quand le projet requiert une constitution de procédures plus complexes".*

Une étude intéressante de MENDELSOHN tente d'ailleurs de faire un parallèle entre les difficultés rencontrées par des enfants de 8 à 13 ans et leur niveau opératoire (en terme piagétien, leur niveau de développement intellectuel) : on donne aux enfants (après un entraînement de 10 heures au LOGO) un quadrillage représentant l'écran et un énoncé de programme que l'enfant est invité à exécuter lui-même en dessinant le trajet de la tortue. Deux types d'erreurs sont isolés :

- les erreurs d'interprétation : erreur sur la signification d'une instruction ;
- les erreurs de coordination : les instructions isolées sont correctement interprétées mais mal agencées ; ex. : certains appliquent à la répétition une sortie de règle de distributivité : l'instruction (répète 4 [av 20 av 90 av 20] est "exécutée" de la manière suivante (répète 4 fois av 20, puis répète 4 fois av 90, etc..

Ces mêmes enfants passent une épreuve de "l'Echelle de Développement de la Pensée Logique" permettant de répartir les enfants en 3 groupes : niveau opératoire concret, niveau préformel, niveau formel. L'étude montre qu'il existe des différences significatives entre les moyennes des erreurs en fonction du niveau opératoire. En particulier de programmes dont la complexité va au-delà de la structure séquentielle semble requérir la mise en place des opérations formelles (10 - 11 ans).

D'autres recherches chez des enfants plus âgés mettent également en évidence des difficultés d'appropriation de certains concepts : l'itération pour des enfants de 11 à 15 ans (LABORDE), la planification et la mise en évidence des implicites chez des élèves de 15 à 16 ans (SAMURCAY).

Enfin, le GRAI (groupe de recherche sur l'apprentissage et l'informatique)[5] poursuit depuis plusieurs années des recherches exploratoires visant à mettre en évidence

(5) Département Education et Technologie, FNDP, 61, rue de Bruxelles, Namur.

les difficultés rencontrées par des adultes lors de leurs premières initiations à l'informatique. Les observations sont réalisées d'une part à l'occasion de nos formations d'enseignants et d'autre part, en situation individuelle avec enregistrement des commentaires et des ordres donnés au clavier, l'adulte étant dans ce cas confronté à une première situation problème LOGO. Les premiers résultats permettent d'épingler un certain nombre de difficultés :

a) La découverte des caractéristiques du dialogue homme-machine semble poser problème. Les adultes observés éprouvent notamment des difficultés à :

   - cerner le type de langage : mode, temps, personne, degrés de simplicité, de complexité de la syntaxe, etc ;
   - cerner les caractéristiques de la tortue : elle a une orientation, etc. ;
   - accepter l'impossibilité de projeter sur la tortue une compétence linguistique ; contrairement à ce qui se passe dans le dialogue humain, les adultes ne peuvent supposer à aucun moment un "background" chez la tortue leur permettant de laisser les implicites habituels dans leurs discours.

b) Un autre problème important est la difficulté de cerner la fonction en cours de la machine : ils confondent ainsi l'exécution et l'édition, l'utilisation du programme et sa construction.

   Les comportements indicateurs de ces problèmes sont les suivants :

   - tenter d'exécuter un programme dans l'édition ;
   - croire que les ordres donnés en mode direct sont enregistrés ;
   - en faisant exécuter un programme, ne pas "jouer le rôle " de l'utilisateur ;
   - attribuer des contenus de variables dans l'éditeur ;
   - etc.

c) La confusion entre le nom de la variable et son contenu est également fréquente : pour certains, il est difficile de savoir quand on la définit et quand on lui donne une valeur ; le problème étant résolu, dans certains cas, en faisant les deux en même temps !

   Ex. : POUR CARRE :20

   Il est à noter que ce problème est en rapport avec celui évoqué au point a) : dans ce cas, l'adulte confond la définition de la procédure et son appel.

   Ce problème se retrouve quand il s'agit de manipuler les variables.

   Ex. : ECRIS "X pour écrire le contenu de X.

d) La distinction n'est pas toujours bien établie entre ce qui est conventionnel et ce qui ne l'est pas. Il n'est ainsi pas rare qu'on nous demande si le nom de telle ou telle procédure ou variable pourrait être modifié.

Tout se passe comme si l'adulte, inhibé par les inévitables erreurs de syntaxe réalisées en début d'apprentissage, semble s'en remettre le plus possible à "ce qui a marché", y compris des désignations purement conventionnelles.

e) Les nombreuses informations contenues dans les messages d'erreurs sont fort peu utilisées ; l'attitude face au "bug" est essentiellement passive (éditer la procédure, la relire), c'est surtout la formulation d'hypothèses et l'organisation de tests qui font défaut.

2. Les problèmes de transfert

En supposant le premier point réalisé, il reste, nous l'avons vu, une question fondamentale : les élèves ont-ils ainsi acquis des techniques générales de résolution de problèmes ?

PEA et KURLAND ont réalisé sur ce sujet une revue de la question intéressante. Ils font d'abord remarquer qu'il s'agit d'une "vieille idée dans un nouveau costume" et que les arguments ressemblent étrangement à ceux utilisés naguère pour la défense de la logique, de la grammaire et du latin.

Passant en revue une série d'études visant à mesurer le transfert réalisé par les élèves des activités de programmation à d'autres types d'activités, ils concluent qu'il n'existe encore que très peu de preuves de bénéfices cognitifs importants. Ainsi des transferts n'ont pu être mis en évidence ni par des mesures de stratégies de résolution de problèmes, ni par des mesures plus proches concernant la rigueur mathématique ou l'acquisition de certains concepts mathématiques.

Ces résultats peuvent paraître décourageants mais il faut cependant insister sur la nécessité d'affiner ce type d'étude dans deux directions au moins :

1) la finesse des mesures de transfert ;

2) la spécification du contexte de l'activité de programmation.

1) Les mesures réalisées sont en effet souvent trop globales (ex. : 5 résolutions de problèmes mesurées en termes d'échec ou de réussite) et trop éloignées des situations LOGO. Des mesures plus fines peuvent faire apparaître des différences significatives parfois dans des domaines a priori peu en rapport avec l'activité de programmation. Un bon exemple nous est donné par l'étude de BIDEAULT visant à mesurer l'impact d'une pratique LOGO sur une épreuve de construction de parcours.

Deux groupes équivalents sont constitués ; le premier seulement prend part, une matinée par semaine, à des activités LOGO. Les deux subissent en fin d'année une épreuve sur feuille de papier : l'enfant dispose d'un plan et d'un bonhomme en plastique ; il lui est demandé de construire un tracé de manière à rejoindre deux maisons sans passer par un certains nombres de points, de rédiger une marche à suivre destinée au bonhomme et enfin de trouver le tracé le plus court.

L'analyse des résultats montre que :

a) Il n'y a pas de différence entre les groupes au niveau du type de stratégie employée.

b) Le groupe expérimental est plus précis dans ses ordres en employant plus souvent la mesure.

c) Le groupe expérimental utilise des stratégies de vérification du codage énoncé en utilisant le bonhomme en plastique.

d) Le groupe expérimental se montre davantage capable d'expliquer le comment de leurs actions. Les expressions verbales utilisées font appel à des comparaisons, à des mesures, etc.

Les bénéfices ne se mesurent donc pas globalement en terme de réussite ou d'échec mais se manifestent par l'acquisition d'attitudes telles que l'utilisation d'un outil de vérification, la nécessité d'expliciter aux autres ses actions, de prouver ses résultats.

2) La question de savoir quels sont les bénéfices cognitifs de la programmation est également trop générale pour une autre raison. En effet, qu'entend-on par programmation ? Qu'a-t-on enseigné aux élèves ? Quel langage ? Combien de temps ? Dans quelles conditions ? Etc.

Il est, en effet, important de préciser le contexte dans lequel les étudiants ont eu l'occasion d'exercer des activités de programmation car celles-ci vont influencer fortement les possibilités de transfert :

> *"La question du rôle du contexte dans l'apprentissage de la programmation est complexe parce qu'il ne s'agit pas d'une compétence unique. Tout comme la lecture elle englobe un grand nombre d'habiletés qui interagissent avec les connaissances antérieures de l'élève, sa mémoire, ses capacités de traitement de l'information, ses stratégies générales de résolution de problèmes telles que l'inférence, la génération d'hypothèses, etc.. L'entraînement à la lecture suppose une gamme étendue d'expériences avec différents genres (narrations, essais, poésie, débat) et avec différents buts. Tout comme la lecture est souvent assimilée à l'habileté de décodage, l'apprentissage de la programmation est souvent équivalente à l'apprentissage d'une syntaxe et d'un vocabulaire d'un langage ; alors que l'entraînement à la programmation, comme la lecture, est complexe et*

> *fortement dépendante du contexte. Nous devons donc commencer par préciser le contexte dans lequel la programmation a été mise en pratique et apprise."* (PEA et KURLAND p.144).

On se rendra compte ainsi que les probabilités de transfert vont fortement varier d'une situation à l'autre. Il est notamment important de savoir :

a) Quel est l'environnement informatique ? Quel langage ? Existe-t-il un éditeur facilitant l'écriture du programme, des aides de debugging, de la documentation, etc. ?

   Ces facilités permettent en effet de rendre le codage plus aisé et de se concentrer ainsi sur les questions plus intéressantes de design, de structuration, d'élégance du programme.

b) Quelles sont les conditions d'enseignement ? Quelles sont les modalités d'accès aux machines ? Les possibilités de sauvetage du travail de l'élève ? Les cours sont-ils centrés sur la syntaxe, le vocabulaire ? A-t-on bien vérifié l'assimilation de concepts fondamentaux (variables - variable globale/locale, etc.) ? A-t-on imaginé des exercices autres que l'entrée des données (ex. : prévoir les contenus des variables à chaque étape de l'exécution - choisir diverses données et prévoir le déroulement du programme en conséquence (avec les "si") ? Quelle est la part de l'enseignement et des exercices ? Quel est le degré de directivité ? A-t-on enseigné des stratégies de debugging ? En LOGO, par exemple, l'enseignement a-t-il été centré sur la découverte libre ou a-t-il été plus structuré et dirigé ?

   PEA et KURLAND montrent ainsi qu'en fonction de ces différentes variables du contexte, les étudiants peuvent se situer à différents niveaux de maîtrise de la programmation. A chacun de ces niveaux, les habiletés transférables vont être différentes.

Niveau 1 "Utilisateur"

L'étudiant est capable d'utiliser des programmes ; on ne peut espérer aucun transfert à d'autres situations mais bien sur son niveau d'alphabétisation informatique. (Exemple : en utilisant différents programmes, il peut distinguer progressivement les fonctions qu'un ordinateur peut prendre en charge de celles qu'il lui est impossible d'assumer.)

Niveau 2 "Encodeur"

L'étudiant connaît la syntaxe et la sémantique des principales instructions d'un langage ; il peut créer des programmes courts sur le modèle de ceux qu'il a déjà rencontré. L'attention est tellement portée sur l'exécution (on cherche à ce que le programme marche), que la probabilité de transfert de stratégies générales est faible.

Niveau 3 "Créateur de programme"

A ce niveau, l'étudiant devient capable de raisonner en termes d'unité de haut niveau ; il connaît des séquences d'instructions réalisant certains sous-objectifs fréquents (lire les informations entrées par le clavier, etc.). Il peut écrire de longs programmes mais peu "conviviaux". Il devient conscient des processus mis en oeuvre dans l'écriture d'un programme et cela semble une des conditions pour qu'un transfert se réalise.

Niveau 4 "Créateur de logiciel"

L'étudiant est alors capable de concevoir des programmes non seulement complexes et structurés mais aussi tournés vers l'utilisateur. Il commente ses programmes, en fait une analyse préalable, et est capable d'en "simuler" l'exécution. C'est à ce niveau que le transfert est sans doute le plus probable : il prend de la distance par rapport au code utilisé et se concentre sur les phases et les processus de résolution du problème que constitue la construction du programme. Par exemple, la nécessité à ce niveau d'être conscient de l'éventail des souhaits des utilisateurs le force à accorder toute son attention à ces derniers. Cette habileté se révèle utile dans beaucoup d'autres situations, en particulier dans les activités d'écriture.

La thèse PEA et KURLAND est que la possibilité d'un transfert n'apparaît que si l'apprenant a atteint un bon niveau de compréhension de ce qu'est la programmation et des habilités qu'elle entraîne. Ce stade ne semble atteint d'une part qu'après un certain temps et d'autre part, qu'à l'aide d'une certaine didactique.

Dans le même ordre d'idée, RIEBER recommande que les recherches sur les transferts à partir des situations LOGO décrivent de manière plus détaillée l'environnement d'apprentissage offert aux apprenants et utilisent principalement cette variable dans la constitution des schémas expérimentaux.

3. Implications au niveau d'une initiation à l'informatique

Après avoir passé en revue quelques problèmes liés à l'apprentissage de la programmation et au transfert, il nous faut en conclure certaines conséquences pratiques au niveau des activités organisées en classe. Que nous disent ces recherches au niveau des contenus et des méthodes d'une initiation à l'informatique ?

1. La première conséquence importante est d'adopter une **position plus modeste et plus réaliste par rapport aux bénéfices escomptés de la programmation.** Plusieurs expériences mettent un terme à certaines illusions pédagogiques concernant l'effet bénéfique automatique de l'ordinateur et mettent l'accent sur l'importance de l'environnement pédagogique.

   Dans le même ordre d'idée, il faut bien resituer le type de "pensée" engagée dans la programmation parmi l'ensemble des processus cognitifs à faire acquérir.

"Ainsi lorsqu'un utilisateur se propose de dessiner une fleur avec le langage LOGO (ou un autre), il se situe toujours dans un registre de concrétisations de capacités assez délimité : celui de l'activité rationnelle par rapport à une fin, de la gestion univoque des moyens, d'une logique non ambiguë, etc."[WEISSBERG p.21].

2. L'objectif étant ainsi plus délimité, la seconde conséquence pratique est **la mise en oeuvre de toute une série de moyens pour que les élèves atteignent une maîtrise suffisante de la programmation** pour qu'un transfert soit possible (niveaux 3 et 4).

   2.1. Une des premières conditions semble être la **limitation de l'âge** à partir duquel il est raisonnable d'envisager une initiation de ce type. En effet, il semble que certaines opérations nécessaires relèvent du stade de développement formel et que, dès lors, ces activités ne seraient accessibles qu'à des enfants de 10-11 ans. HOC rappelle ainsi que si l'activité de programmation partage les caractéristiques des activités de résolution de problème, elle exige également que l'apprenant explicite les procédures utilisées.

   DUCHATEAU établit une distinction similaire entre le "faire" (résolution de la tâche) et le "faire faire" (faire résoudre le problème par un exécutant, ce qui nécessite une mise à plat et une explicitation consciente des étapes de la résolution de la tâche).

   Une maîtrise du niveau 4 requiert de plus une capacité de réflexion plus générale sur les mécanismes eux-mêmes de résolution de problèmes. Ainsi en LOGO, la maîtrise de la récursivité passe par la découverte de ce qu'est une approche récursive d'un problème, de son utilisation assez rare en situation naturelle, des nouvelles manières de penser qu'elle introduit.

   L'étudiant - comme nous l'avons vu plus haut - dispose, à ce stade, de plans, de schémas pour certains buts spécifiques, "intériorise" les stratégies utilisées pour construire les programmes antérieurs. Ces opérations se rapprochent de ce que les anglo-saxons appellent la "métacognition". De nombreuses recherchent ont montré que ces capacités de réflexion sur les mécanismes de cognition ne se développent pleinement qu'à l'entrée de l'adolescence.

   2.2. **Les cours doivent porter autant sinon plus sur les méthodes que sur les contenus.** Nous avons vu ci-dessus qu'une énumération des instructions et de la syntaxe du langage était loin de suffire. Par exemple, une période de travail sur papier doit précéder ou accompagner le travail sur machine : elle consistera par exemple en un exercice de décomposition de problèmes, en une élaboration, à partir des expériences de chaque étudiant, d'une stratégie de recherche d'erreur, etc.

Il serait aussi intéressant de rassembler les règles, "trucs" et stratégies que les experts en programmation utilisent couramment (ex. : insérer dans un programme une instruction qui permette d'afficher le contenu des variables à certains points stratégiques du programme).

2.3. **Les cours devraient être centrés sur la résolution de problèmes** - puisque c'est fondamentalement cela qu'il s'agit de transférer - par les élèves plutôt que d'être un enseignement de contenus. Pourquoi d'ailleurs ne pas **intégrer cette initiation informatique aux autres disciplines** plutôt que d'en faire une matière à part, que ce soit dans l'enseignement fondamental ou dans l'enseignement secondaire. En effet, s'il est évident que les tâches sur lesquelles portent les premiers essais de programmation doivent être simples, on ne voit pas pourquoi elles devraient être sans signification et non pertinentes : elles pourraient ainsi être en rapport avec la vie quotidienne des élèves ou avec des questions apparues à l'occasion d'autres cours.

L'intégration de la programmation au cours de mathématique semble, par exemple, enrichissante (JONES)[6]. Ainsi une compréhension en profondeur de certains concepts (ex : le concept de "premier") peut être amenée par la construction d'un programme (ex : programme qui génère des nombres premiers). Ce raisonnement semble cependant mieux s'appliquer à la découverte de logiciels ou d'outils informatiques qu'à la programmation : celle-ci nécessite en effet une longue période d'entraînement notamment aux caractéristiques du langage et l'on voit mal cet apprentissage prendre place à l'intérieur de disciplines dont le programme est par ailleurs déjà surchargé.

2.4. Une dernière conséquence porterait d'une part sur l'intensification des recherches sur les processus sous-jacents à la programmation et d'autre part sur la **mise au point d'outils d'aide à la conceptualisation** de notions que ces recherches auraient isolées comme des points cruciaux dans la maîtrise de la programmation.

Plusieurs chercheurs ont ainsi montré que la source de certaines difficultés résidait dans une mauvaise représentation du fonctionnement de la machine (DU BOULAY, MAYER)

MAYER, par exemple, fait l'hypothèse que les étudiants apprennent un langage de programmation en reliant le rôle des instructions à des modèles physiques ou mécaniques. D'où l'idée de fournir aux étudiants un modèle concret du fonctionnement de la machine (au niveau des instructions du langage

(6) Mais rappelons qu'il n'y a aucun lien privilégié : on peut ainsi l'intégrer au primaire, dans les activités d'éveil de manière à traiter, par exemple, les résultats d'une enquête ; l'essentiel étant de résoudre par la programmation, ou plus généralement par l'informatique, comme nous le verrons dans la remarque finale, un problème posé dans la classe.

seulement)[7] leur permettant d'intégrer et de mieux assimiler ces notions, en se représentant, à l'intérieur d'un modèle simple et visuel, le rôle des diverses instructions.

MAYER montre que les étudiants qui disposent de ces modèles assimilent mieux les diverses instructions vues dans la suite ; leur score à des tests de transfert (à d'autres domaines informatiques proches) est également plus élevé.

DU BOULAY a également mis au point un modèle semblable à celui présenté ci-dessus pour l'apprentissage du LOGO ; ce modèle permet à l'apprenant de "jouer à la machine" (placer des données dans l'espace de travail, lire au clavier, etc.). Il oppose ainsi deux approches de l'apprentissage de la programmation : "l'approche boîte noire" dans laquelle les mécanismes utilisés par l'ordinateur doivent rester cachés à l'utilisateur et "l'approche boîte de verre" dans laquelle l'effet de chaque instruction sur ce qui se passe dans la machine est décrit au moins de manière métaphorique.

De nombreuses autres stratégies et dispositifs ont déjà été définis et testés ; citons part exemple :

- l'utilisation de métaphores (cf. métaphore de "l'exécutant comme gestionnaire de casiers" (DUCHATEAU)) ;
- la mise au point de minilangages rudimentaires et adaptés aux apprentissages visés (ex. : définition de compétences limitées d'un robot pour débutants en LOGO (LABORDE) , ELOGO (DU BOULAY) ;
- illustration du fonctionnement d'un programme : un pointeur montre le déroulement de l'exécution, les contenus actuels des variables apparaissent à tout moment (BIP de BARR, BEARP et ATKINSON) ; le jeu de primitives chargeables "DEBUG" dans le LOGO LCSI sur le Macintosh permet une représentation similaire) ;
- livre ordinateur (MUPY ; exercice, b.d., petit ordinateur simplifié) ;
- la mise au point d'interventions explicites pour favoriser les activités de planification : DALBEY a ainsi construit et testé une série de cinq leçons visant à encourager les étudiants à donner plus d'importance à l'étape intermédiaire entre la définition des spécifications et le codage du programme. Il utilise à cette fin une méthode de représentation graphique de l'organisation logique du problème ("structure diagramming") ;
- le jeu de rôle dans lequel les différents enfants prennent en charge des fonctions spécifiques de la machine de manière à se représenter ce qui se passe dans la machine lors de la programmation (BEDENES).

---

(7) Il ne s'agit pas de décrire le fonctionnement du hardware mais de proposer une "notional machine", c'est-à-dire une machine simplifiée, idéalisée dont les propriétés dépendent des concepts utilisés par le langage.

- fournir un modèle de base du fonctionnement d'un appareil informatique (VENTURINI).
- réalisation d'un robot par une équipe d'élèves (MYX).

D'autres moyens restent à inventer pour faciliter la maîtrise de ces outils informatiques.

Une dernière remarque : nous avons essentiellement parlé jusqu'à présent de programmation ; or, il apparaît de plus en plus clairement que d'autres activités informatiques sont également enrichissantes au point de vue cognitif. L'utilisation de progiciels (traitement de texte, gestion de fichiers,etc.) notamment semble se révéler un excellent moyen pédagogique en tant qu'outils d'entraînement au traitement d'information (cueillette, mise en commun, organisation, analyse et communication des informations).

Un exemple de ce type d'applications est le projet de l'équipe de JAN PALKIEWICZ de développer par l'utilisation de banque de données la pensée formelle d'adolescents. La consultation future les encourage à :

1. bien analyser leur problème, le transformer en concept clé et établir des relations entre eux ; établir des hypothèses ;
2. comparer leur résultat avec le thésaurus ; améliorer leur définition conceptuelle ;
3. définir des stratégies de recherche .

Rarement sans doute une discipline s'est introduite dans le curriculum de l'enseignement général avec autant de rapidité que ne l'a fait l'informatique. A côté de la pression sociale sans doute déterminante, cette percée a probablement été facilitée par l'annonce d'objectifs de haut niveau tels que le développement de la pensée logique.

Nous avons voulu montrer dans cette communication que si ce cours veut tenir ses promesses un certain nombre de difficultés devront être résolues et des environnements d'apprentissage adéquats devront être construits. En particulier, un défi important consiste à savoir comment amener, dans le temps, habituellement limité, imparti à ce cours, les apprenants à un degré de maîtrise des principes de la programmation suffisant pour qu'un transfert puisse se réaliser.

BIBLIOGRAPHIE

BEDENES J.Y., *Simulation*, Bulletin de l'A.C.I.E., 2, mai 1985.

BIDEAULT A., *Procédures d'enfants de CE2 dans une tâche de construction de parcours*,Enfance, 2-3, 1985.

DALBEY J., TOURNIAIRE F. ET LINN M.C., *Making programming instruction cognitively demanding* : an intervention study, Journal of Research in Science Teaching, Vol. 23, n°5 (pp.427-43).

DU BOULAY B. AND OSHEA T., *Teaching novices programming*, in Computing Skills and the User Interface edited by M.J. COOMBS and J.L. ALTY, Academic Press, 1981.

DU BOULAY B., OSHEA T., MONK J., *The black box inside the glass box* : presenting computing concepts to novices, Int. I. Man-Machines Studies, 14, 1981.

DUCHATEAU CH., *Programmer !*, Wesmael Charlier, 1983.

HOC J.M., *L'étude psychologique de l'activité de programmation*, Technique et Science Informatique, Vol. I, n°5, 1982.

JONES K.L. ET LAMB C.E., *Programming in mathematics education* : a essential component in the developpment of reasoning skills, Computers in education, K. DUNCAN and D. HARRIS (eds) Elsevier Science Publishers B.V., 1985.

LABORDE C., BALACHEFF N., MEJIAS B., *Genèse du concept d'itération* : une approche expérimentale, Enfance, n° 2-3,1985.

MAYER R.E., *The Psychology of How Novices Learn Computer Programming*, Computing Surveys, Vol. 13, n°1, mars 1981.

MENDELSOHN P., *L'analyse psychologique des activités de programmation chez l'enfant de CM1 et CM2*, Enfance, n° 2-3,1985.

MYX A., *Spécial robotique*, Bulletin de l'A.C.I.E., 3, mai 1985.

PALKIEWICZ J., *Développement de la pensée formelle des adolescents par l'utilisation systématique de banques de données*, Congrès Cognitiva 85, Paris, 4-7 juin 1985.

PEA R.D. AND KURLAND D.M., *On the cognitive effects of learning computer programming*, New Ideas in Psychology, N. Y., Pegamon Press, Vol. II, 2, 1984.

RIEBER L.P., *Logo and its promise* : A Research Report, Educational Technology, février 1987.

SAMURCAY R., ROUCHER A., *De "faire" à faire faire"* : planification d'actions dans la situation de programmation, Enfance, n° 2-3, 1985.

VENTURINI P., *Un premier modèle de fonctionnement d'un appareil informatique*, Bulletin de l'A.C.I.E., 2, mai 1985.

WEISSBERG J.L., *La maladie infantile de l'informatique*, Education Permanente, 70-71, 1983.

ANNEXE

Résolutions de la C.N.A.P. (a) ;

Orientations fondamentales du Conseil Central de l'Enseignement Primaire Catholique (b) ; Conclusions du colloque Education et Société du 3ème type, Club Athena, Technologies-Education (c) ;

Proposition de développement - Ministère de l'Education, Québec(d) ;

Note du Directeur des Ecoles en France (e) ;

Programme d'études : Introduction à la science de l'informatique (Ministère de l'Education du Québec, septembre 1982) ; Commission Bündlander sur la planification de l'éducation et l'aide à la recherche, décembre 1985. (Projet pour l'introduction d'une formation aux techniques de l'information à l'école).