i.r.e.m.
UNIVERSITE PARIS VII

ACQUISITION DE SAVOIRS ET DE SAVOIR-FAIRE
EN INFORMATIQUE

Par J. ROGALSKI

cahier de didactique des mathématiques

numéro 43

MAI 87

Acquisition de savoirs et de savoir-faire en informatique

Janine Rogalski
Chargée de recherche CNRS
Laboratoire de Psychologie du développement et de l'éducation de l'enfant
46, rue Saint-Jacques. 75005 Paris.

Introduction

Le problème des acquisitions en informatique en termes de savoirs et de savoir-faire est évidemment lié à la définition des objectifs mêmes d'un enseignement en la matière.

Son approche appelle des interrogations sur l'informatique en tant que contenu de connaissance et en tant que pratique; elle pose aussi la question difficile de l'évaluation des effets produits sur des élèves par un enseignement donné.

Résoudre ce problème nécessite, entre autres, de dégager ce qui relève de caractères internes au domaine lui-même, ce qui a été produit par l'enseignement et ce qui est attaché à des caractères propres à l'élève lorsqu'on observe des acquisitions ou au contraire des difficultés.

Enfin, du point de vue de l'enseignement, avancer dans la solution de ce problème vise à pouvoir mieux construire et choisir des situations d'enseignement, envisagées dans la durée, candidates à répondre aux objectifs retenus.

Les réflexions qui suivent visent à apporter quelques éléments au débat en ce qui concerne l'enseignement de la programmation à des élèves du second cycle de l'enseignement secondaire, dans le cadre d'un enseignement institutionnel.

Deux voies possibles de constitution de nouvelles connaissances

D'observations empiriques dans le domaine scolaire et extra-scolaire on peut faire l'hypothèse que deux voies existent pour l'acquisition de nouvelles connaissances, bien qu'elles soient probablement très inégalement "empruntées".

Elles se distinguent par le statut de la signification dans le processus de construction de connaissances. Notre hypothèse est qu'elles peuvent intervenir l'une et l'autre de manière "locale" dans les acquisitions d'un champ conceptuel donné.

"Le fonctionnement crée du sens":

Le fonctionnement en lui-même des éléments d'un système inconnu peut permettre la constitution d'une signification des notions et des règles de ce système, pour la personne qui le fait

fonctionner. Au point de départ ces règles de fonctionnement sont sans signification autre qu'interne au système, et locale.

Par exemple, on sait que les joueurs d'échecs, à partir d'une pratique de ce jeu, dépassent la connaissance en termes de règles de déplacement des pièces (éléments de départ dans l'initiation) pour attribuer une signification à un ensemble de dispositions sur l'échiquier, en termes de valeur stratégique, connaissance décisive pour devenir un joueur expert.

On a l'expérience empirique d'élèves assimilant (et s'appropriant effectivement) des notions algébriques sur la base initiale d'un jeu de formes sans signification (celles des écritures algébriques): la constitution de sens en algèbre à partir de règles d'écriture peut relever d'un processus proche de celui du joueur d'échecs. On a pu observer aussi des étudiants qui intégrent, assimilent les concepts d'une théorie à partir d'une présentation formelle, sur la base du fonctionnement d'un système axiomatique, qui peuvent non seulement leur donner un sens mais les mettre ensuite en relation avec d'autres domaines : le même processus de création du sens à partir du foncitonnement nous semble à l'oeuvre à un niveau beaucoup plus complexe.

Empiriquement on sait que le temps nécessaire à la constitution de savoirs opérationnels construits par une telle voie est long, et on connait l'échec de très nombreux élèves à en bénéficier dans le cadre scolaire. On doit donc s'interroger sur les conditions concernant le "passé cognitif" de l'élève , son mode d'approche du savoir, ainsi que sur le temps d'activité qu'il est nécessaire de consacrer au contenu considéré, pour qu'une telle voie de construction de connaissances soit effective. On doit aussi s'interroger sur le type de fonctionnement qu'il faudrait organiser pour optimiser un tel processus de création de sens.

Nous prendrons donc comme point de départ pour une initiation large à l'informatique l'hypothèse que l'enseignement ne peut prendre le risque de se fonder sur le fonctionnement de tels modes d'acquisitions de la part de l'ensemble des élèves au début du second cycle de l'enseignement secondaire (voire au-delà). Toutefois nous n'excluons pas l'hypothèse que, localement, un tel mode d'acquisition puisse intervenir. (L'introduction à certains logiciels sous forme de règles d'utilisation, exposées dans un manuel d'utilisateur, conduit bien à des acquisitions relevant de ce mode, toutefois notre hypothèse est qu'elle s'appuie souvent implicitement sur l'existence de systèmes de représentations et de

traitements (Hoc) "spontanés" suffisamment voisins pour introduire du sens au-delà des règles d'utilisation).

"La construction par accommodation/assimilation"

Une autre voie est la construction de nouvelles connaissances à partir de celles de champs conceptuels déjà appropriés . Cette voie se distingue de la précédente dans la mesure où la nouvelle connaissance prend appui à l'extérieur de son champ propre, dans des acquis antérieurs. Les problèmes qui nécessiteront de nouveaux concepts pour les résoudre peuvent être abordés avec les outils des champs conceptuels déjà appropriés et peuvent donc avoir du sens pour l'élève.

Dans la construction de nouveaux savoirs, des notions ou des représentations appartenant à un domaine de connaissance préexistant peuvent jouer le rôle de précurseurs. Cela peut aussi être le cas de modes de résolution, voire de méthodes d'approche de problèmes. Nous allons en donner quelques exemples concernant des concepts de base en informatique.

La variable a comme précurseur possible la variable mathématique rencontrée antérieurement dans l'enseignement et qui s'intègre dans un champ conceptuel plus ou moins large. L'existence de ce précurseur permet à l'élève de commencer à donner directement du sens à la variable informatique , mais elle peut rendre difficile la représentation dynamique de cette dernière et risque de réduire ultérieurement la notion de fonction à celle de fonction numérique, presque exclusivement rencontrée dans les problèmes de mathématiques.

De même l'existence de la représentation symbolique "=" de l'égalité des variables numériques joue un rôle producteur dans les acquisitions initiales des tests en programmation. Pour certains élèves cette signification se transfère sans problème aux tests sur des variables non numériques, par exemple les chaines de caractères, avec utilisation du signe d'égalité; d'autres élèves transposent les règles d'action habituelles pour ces types de variables c'est à dire qu'ils les traitent avec le langage naturel; le symbolisme qu'il faut utiliser en programmation joue alors pour eux un rôle d'obstacle à l'écriture de procédures conditionnelles. De plus la familiarité même de ce symbolisme peut rendre difficile le changement de point de vue qui consiste non pas à comparer la valeur d'une variable A à une autre valeur B mais à tester si une certaine propriété est vraie (la valeur logique de la relation A=B

est: VRAI).

L'activité de modélisation de situations a elle-même déjà été rencontrée par des débutants en programmation à propos des mathématiques , et en particulier de l'algèbre. Il peut en être de même pour l'écriture de relations (x=a, y>3..) ou même pour la structuration des données (arbres, tableaux..). Tout cela fait partie de précurseurs possibles pour les acquisitions en programmation; ce qui rend compte du fait que des élèves à l'aise en mathématiques vont en général plus vite dans les acquisitions initiales.

Nous voudrions souligner que des précurseurs ne sont pas des prérequis: ils peuvent être utiles pour l'acquisition des nouvelles connaissances sans être pour autant nécessaires, d'autant qu'il peut y avoir plusieurs précurseurs différents pour une notion donnée.

Rôle producteur et réducteur des précurseurs

Les précurseurs ont un double rôle. Ils contribuent à donner du sens pour l'élève à l'utilisation de concepts nouveaux, ou facilitent la manipulation de représentations symboliques nouvelles: c'est un rôle **producteur**. Ils peuvent aussi être des obstacles si l'élève en transpose des caractères non adéquats: il s'agit là d'un rôle **réducteur**.

On a vu ce dernier point dans le cas des variables, dont le caractère statique en mathématiques peut constituer un obstacle à la représentation de la modification possible de la valeur d'une variable lors de l'éxécution d'un programme.

Les procédures de résolution familières, "à la main", à côté d'un rôle producteur fréquent, peuvent également jouer un rôle réducteur insidieux, souvent sous-estimé. Ainsi, par exemple, le caractère familier de la procédure de calcul de la somme des n premiers nombres cache l'invariant qui permet de construire de façon fiable le programme itératif correspondant.

Dans certains cas, les procédures familières ne sont même d'aucune utilité. Ainsi dans la division euclidienne c'est la relation fonctionnelle "a=bq+r et r<b" qui permet l'écriture du programme (mais elle n'est guère familière aux élèves) alors que l'algorithme de division bien connu dans sa pratique dans la numération décimale met en oeuvre des processus très complexes,

dont l'identification et la traduction procédurale sont beaucoup plus difficiles.

On voit que l'analyse fine des divers éléments qui peuvent servir de précurseurs, sur lesquels s'appuyer pour faire comprendre aux élèves la signification de nouvelles notions, ou de nouvelles écritures et donner du sens aux problèmes qui exigeront ensuite l'introduction de nouveaux concepts propres à l'informatique, est tout à fait nécessaire pour jouer du rôle producteur de ces précurseurs en "neutralisant" au maximum leur rôle réducteur.

On retrouve dans cette dialectique des rôles réducteur et producteur des précurseurs le problème des **dépassements**: le précurseur qui ,avec les règles d'action qui lui sont associées dans son domaine d'origine, fournit des outils immédiatement utilisables pour agir dans le nouveau domaine, doit être transformé pour interagir de manière cohérente avec les nouveaux objets et outils du nouveau domaine de savoir.

Le moteur de la construction de nouveaux savoirs est alors l'activité de l'élève lorsqu'il opère sur de nouveaux problèmes, pour lesquels l'enseignement apporte des outils cognitifs nouveaux, qui pourront d'ailleurs modifier par leur mise en oeuvre le sens initial du problème. Une réalisation d'un tel processus a été proposée par Dubinsky (Dubinsky, 1987) pour opérationaliser les concepts constituant le raisonnement par récurrence à l'aide d'un langage de programmation adéquat.

Spécificités de l'informatique comme objet d'enseignement.

Un objet d'enseignement met en cause une interaction entre un contenu à enseigner et les élèves chez lesquels on veut que se construise une certaine connaissance opératoire. Par connaissance opératoire nous entendons un ensemble de savoirs et de savoir-faire, organisé en champs conceptuels avec des règles d'action, qui permette au sujet de résoudre un champ de problèmes.

Les spécificités de l'informatique doivent donc être envisagées du point de vue du contenu à enseigner et du point de vue du sujet qui doit acquérir des connaissances.

Spécificités du point de vue du contenu

L'informatique peut être considérée sous son versant de domaine scientifique avec ses concepts propres; elle peut aussi être considérée sous le versant des pratiques qui lui sont liées.

Fondamentalement, et encore plus que les mathématiques,

l'informatique est une "discipline de service": elle apporte à divers domaines, dont l'enseignement fait d'ailleurs lui-même partie, ses outils théoriques et techniques pour contribuer à la résolution des problèmes qui y sont posés.

Nous parlons d'outils théoriques lorsque les concepts informatiques sont mis en oeuvre comme des outils, des intermédiaires, pour résoudre les problèmes d'un autre domaine. Les outils techniques concernent tout l'environnement informatique; ils sont utilisés lors de la réalisation effective des solutions apportées avec les outils théoriques.

C'est cet aspect de l'informatique comme outil de pensée qui a été mis en avant dans l'introduction institutionnelle d'un enseignement d'informatique dans le second cycle des lycées. L'initiation de culture générale à l'informatique implique ainsi une part importante de modélisation de situations-problèmes.

L'activité de modélisation est certes également un élément-clé dans les mathématiques ; mais la modélisation informatique présente deux spécificités. D'une part les concepts de représentation et de traitement des données sont particuliers et la structuration nécessaire au passage entre données et résultats est en interaction avec les langages utilisés, ainsi que les méthodologies d'approche des problèmes à programmer. D'autre part il faut souligner une spécificité qui est celle de l'exigence de réalisation effective; on attend des solutions apportées par l'informatique qu'elles soient effectives : les opérations impliquées par la solution doivent non seulement être calculables mais exécutables dans un temps limité. Certes d'autres modélisations n'ont d'intérêt que par leur exécutabilité, mais ici il faut insister sur l'extraordinaire développement du champ théorique qui est lié à des problèmes en apparence essentiellement, sinon exclusivement, techniques.

Le caractère exécutable n'est donc pas seulement d'ordre logique : l'identification même de la situation finale recherchée prend en compte sa réalisation matérielle (fichier informatique, listage, écriture sur écran, sortie analogique...).

Aux spécificités que nous venons d'évoquer s'ajoute évidemment l'existence de concepts et de méthodes propres à l'informatique. Leurs caractéristiques comportent des interactions particulières entre logique, langage, technique, qui conduisent, entre autres, à des formes propres de modélisation, ne se réduisant pas à des modèles mathématiques (ou logiques).

On peut définir très schématiquement trois niveaux différents, non indépendants, pour le découpage de l'informatique en grands domaines conceptuels. La figure 1 donne une représentation de ce découpage. Ces niveaux correspondent à l'existence de types différents de manuels et d'enseignements: le découpage effectué est de nature sans doute plus didactique qu'épistémologique.

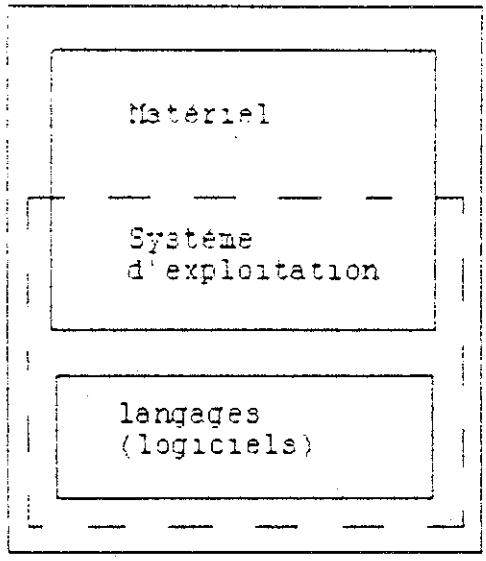

Figure 1

Même en se limitant au domaine de la programmation, les filiations possibles avec des notions mathématiques ne réduisent pas la programmation à une activité mathématique. En effet on ne peut analyser une notion isolée, mais il faut prendre en compte ses relations avec d'autres notions et les opérations spécifiques dans lesquelles elle intervient.

Or les relations entre les notions informatiques diffèrent des relations entre notions mathématiques voisines. Ainsi la notion de variable informatique intervient, dans une large classe de langages, en liaison avec l'instruction d'affectation qui ne s'identifie pas à l'égalité définitionnelle des mathématiques ; les fonctions informatiques dans lesquelles elles interviennent possèdent des traits spécifiques (existence d'effets de bords) qui nécessitent de différencier des variables locales et des variables globales.

Même dans un langage comme PROLOG, étroitement lié à la logique dans ses concepts et sa syntaxe, le processus d'instanciation des variables introduit des spécificités liées à l'exécution du programme (ainsi la conjonction de propositions peut ne plus être commutative lorsque l'une d'entre elles est une négation de proposition).

Spécificités du point de vue des élèves.

Par ailleurs, du point de vue des élèves, l'informatique présente des spécificités qui influent sur les processus d'acquisition et d'enseignement.

Tout d'abord il n'y a pas pour l'informatique le même arrière-plan d'activités intellectuelles "spontanées", conduites par l'enfant indépendamment de toute volonté didactique extérieure, comme c'est le cas si on considère les premières notions numériques ou spatiales. Cela est du moins la situation actuelle où l'ordinateur n'est pas un objet familier sur lequel l'enfant exercerait librement des activités, explorant le "monde" informatique, et se poserait des problèmes impliquant des notions informatiques.

De plus l'informatique comporte en tout état de cause un savoir humain considérable, non seulement comme objet matériel fabriqué (comme artéfact, selon la terminologie des psychologues étudiant les processus de formation de concepts) mais dans les fonctions remplies au travers de logiciels.

Les précurseurs sur lesquels on peut appuyer une initiation ne sont donc pas à chercher dans des notions spontanément acquises antérieurement à l'enseignement, mais doivent être recherchés dans d'autres domaines.

Par ailleurs les types de représentations symboliques et de représentations mentales nécessaires à la maitrise de la programmation, c'est à dire à sa pleine efficacité, ne semblent pas être celles qui se construisent dans l'action ou qui en reflètent le caractère dynamique. Même dans des langages dits impératifs ou procéduraux la construction de programmes logiquement corrects implique le recours à des représentations en termes de situations ou d'états , représentations qui ont un caractère statique.

La maîtrise de la dialectique entre exécution d'actions, avec son caractère dynamique (et la séquentialité de cette exécution) et succession de situations, avec une analyse statique de propriétés, apporte une complexité que les élèves ne rencontrent que plus tard (et pas tous) avec certains concepts mathématiques comme les suites, par exemple. (Cela constitue une spécificité par rapport à l'enseigement de l'informatique lorsqu'on initie à la programmation des élèves de seconde.). Mais, dans tous les cas, l'existence d'une exécution avec ses contraintes de déroulement temporel donne un statut spécifique à ce problème de coordination des deux points de vue dynamique et statique.

Intervention des représentations mentales du dispositif informatique dans les acquisitions.

Toute résolution de problème par l'informatique implique la prise en compte de l'existence d'un dispositif informatique: ordinateur avec ses périphériques, interfaces logicielles et matérielles, langage utilisé etc...

On peut représenter par le schéma de la figure 2 la situation de résolution d'un problème par l'informatique. Il peut s'agir d'une tâche de programmation mais aussi bien d'un traitement de données par un logiciel comme un gestionnaire de données ou un tableur, ou d'une production graphique avec un logiciel d'aide à la conception.

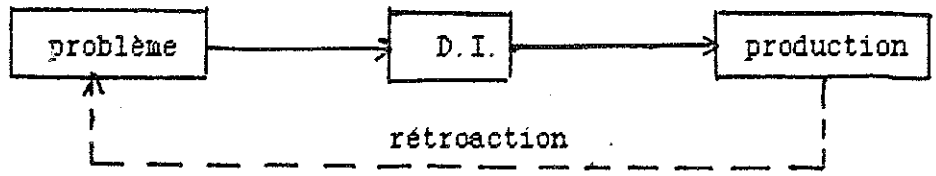

figure 2

Dans une telle situation de résolution les élèves sont contraints d'interpréter la production effective d'une solution non seulement du point de vue de sa logique interne mais aussi de la représentation qu'ils se font du fonctionnement du dispositif informatique au travers duquel a lieu la réalisation de la solution.

M. Artigue nous a fait souligner qu'on trouvait là un phénomène de "distanciation" produite par un dispositif technique analogue à celui produit en physique par l'usage d'un appareil.

Lorsque le problème consiste en l'écriture d'un programme, le sujet doit anticiper le rôle du dispositif informatique lors de l'exécution du programme. (Par exemple une instruction d'écriture à l'écran de résultats ne garantit pas la lisibilité par l'utilisateur ne serait-ce qu' à cause du "défilement" de l'écran).

Ainsi, en écrivant, en analysant ou en complétant un programme, les élèves doivent mettre en oeuvre, même inconsciemment, des représentations sur les relations fonctionnelles entre le programme écrit, le dispositif informatique par lequel il sera exécuté et l'opérateur qui va utiliser le programme.

Des représentations fausses ou incomplètes vont interférer avec le processus d'algorithmisation, lors de l'écriture du programme ou dans l'analyse ou la prévision d'une exécution. Par exemple, les observations de classe ne manquent pas sur les difficultés des élèves à maitriser les instructions de lecture et d'écriture .

La confusion des rôles entre ordinateur et utilisateur apparait comme un niveau initial, en général fugace, de fausses représentations des relations entre programme, ordinateur, et utilisateur lors de l'exécution d'un programme.

On peut en trouver trace dans l'écriture de AFFICHER 'aller en 3' (message à l'utilisateur) au lieu de ALLER EN 3 (instruction pour l'ordinateur).

Un niveau ultérieur de représentations erronées est celui où les élèves attribuent à l'ordinateur les mêmes compétences de compréhension qu'à un interlocuteur humain. On a pu observer après plus de 16 heures d'enseignement de la programmation (en LSE) que près d'un élève sur deux (dans des classes de seconde des lycées) pouvait faire des erreurs relevant de cette représentation.

Ayant à prévoir ce qui se passerait si dans un programme qu'ils avaient écrit (en LSE) l'utilisateur entrait "BOF" comme réponse à une question (posée juste avant la ligne d'instruction de fin du programme) réclamant dans son libellé une réponse OUI ou NON (avec un traitement de reprise du programme pour la réponse négative), ils ont proposé l'arrêt de l'ordinateur, l'affichage de messages d'erreurs ou de point d'interrogation ; les justifications avancées par certains étant du type: "l'ordinateur qui s'attend à une réponse précise va se bloquer".

Un des caractères du fonctionnement d'un dispositif informatique sur lequel nous voudrions insister est celui de la séquentialité des opérations lors de l'exécution d'un programme.

Or, pour un interlocuteur humain, d'une part des informations peuvent être précisées après un début d'utilisation, d'autre part l'existence de présuppositions dans le fonctionnement naturel de la langue peut amener à supprimer certaines informations dans le discours ou ne pas spécifier leur organisation temporelle.

Dans des problèmes d'écriture ou de complétion de programmes conditionnels avec plus de deux cas à gérer, nous avons observé de nombreux élèves dont les réponses étaient compatibles avec l'utilisation d'une forme de présupposition (implicite): "dans la partie de programme concernant les traitements conditionnels, si

une condition A a été traitée les autres instructions s'appliquent au cas **non** A réciproquement, tant que le traitement de la condition A n'est pas achevé les instructions s'appliquent au cas A".

Si dans un langage comme micro-PROLOG la question de la séquentialité ne se pose pas en ces mêmes termes elle n'en existe pas moins dans les problèmes d'ordre d'évaluation. Ainsi dans leur manuel sur micro-PROLOG, Clark et McCabe insistent sur la nécessité de connaitre l'ordre d'évaluation des conditions pour rédiger certaines questions conjonctives ou comportant une négation.

Identifier les raisons d'erreurs ou d'échecs d'élèves qui relèvent de ces questions de représentations peut accélérer la disparition de ces erreurs, ou donner à l'élève des moyens de les repérer et de les réparer. Cela peut contribuer à éviter que certains élèves ne "décrochent" parce que le niveau d'explication en termes d'analyse de la situation ou en termes de structure logique ou de syntaxe ne correspond pas à celui de leur dysfonctionnement.

Nous avons développé ces derniers points car ils sont un exemple des problèmes posés par des caractères spécifiques de l'informatique lors des acquisitions, sans pour autant les recouvrir tous.

Peut-on, doit-on enseigner des méthodes?

Nous avons considéré plus haut l'informatique à la fois comme un domaine de savoir avec des notions à acquérir et comme moyen théorique et pratique de modélisation. Cette dernière activité implique la maîtrise de notions informatiques mais ne s'y réduit pas.

L'expérience du travail professionnel a montré qu'il s'agissait d'une activité complexe dont la gestion par les professionnels eux-mêmes devenait souvent peu efficace pour des problèmes d'une certaine complexité et d'une certaine ampleur. Il est apparu dans ce cadre la nécessité d'une formation de type méthodique. Ainsi enseigne-t-on à des étudiants en formation d'informatique de gestion une méthode institutionnelle et fortement codifiée (LCP pour Logique de Construction de Programmes).

Dans l'enseignement universitaire plusieurs méthodes sont également enseignées, qui ont en commun de distinguer le passage du problème à l'algorithme , c'est à dire une description logique du

schéma du passage conduisant des données aux résultats (phase souvent appelé analyse), et le passage de l'algorithme au programme, c'est à dire la "traduction" de l'algorithme dans un langage acceptable par le système informatique sur lequel on travaillera. La méthode est explicitée essentiellement en ce qui concerne la phase d'analyse. (Deux chapitres sont intitulés "discours de la méthode" dans le livre de A.Ducrin; ils comportent respectivement 17 pages pour l'analyse, 5 pages pour la "traduction").

Il se pose pour l'enseignement général de l'informatique la question de la place à donner à la formation éventuelle à des méthodes pour l'acquisition des outils théoriques et techniques nécessaires à la modélisation.

Il faut rappeler ici ce que nous entendons par outils théoriques et techniques: les contenus du domaine sont considérés dans le rôle qu'ils jouent dans la résolution de problèmes (avant de l'être comme objets sur lesquels s'interroger). Ils comportent à la fois les notions (celle de structure conditionnelle par exemple, comme outil théorique pour l'organisation d'un programme résolvant un problème) et la façon dont ces notions se réalisent dans un langage de programmation donné. La distinction est évidemment très voisine de celle d'analyse et de transcription dans un langage, mais nous ne voulons pas préjuger ici de l'indépendance entre ces deux phases.

Ainsi pour modéliser des situations dans lesquelles le traitement va dépendre de certaines conditions, les structures conditionnelles sont nécessaires en tant que schémas de contrôle général (c'est alors un "outil théorique"); pour rendre la solution exécutable il faut l'exprimer dans un langage effectif de programmation : la syntaxe des structures condtionnelles dans ce langage va être plus ou moins adaptée aux différents types de problèmes condtionnels (les structures conditionnelles explicitées dans un langage sont alors des "outils techniques").

La question d'une formulation libre de langage et directement traitée par un dispositif informatique est ouverte dans les recherches sur les aides à la programmation; elle ne se pose pas à l'heure actuelle pour le niveau d'enseignement étudié dans le cadre de l'alphabétisation informatique. En tout état de cause cette question pose le problème du statut du langage dans la modélisation (l'utilisation de langages algorithmiques généraux pose de toutes façons le problème de leur adaptation aux problèmes abordés et de

leur appropriation par les programmeurs.).

En ce qui concerne les méthodes, elles visent à expliciter ce qu'il y a de commun dans une activité de modélisation qui soit efficace (et conduise à des algorithmes et des programmes corrects). Cette question se pose à différents niveaux: organisation d'un problème complexe, modélisation d'une situation "simple". Nous allons l'illustrer dans un tel cas simple.

Pour modéliser des situations qui ont comme caractère commun l'existence de traitements conditionnels, la question se pose pour des élèves débutants de "commment utiliser l'outil théorique de structure condtionnelle". Il peut apparaitre utile d'organiser systématiquement les cas possibles pour assurer exhaustivité et unicité de leur traitement. Il peut être utile de considérer les différentes représentations graphiques possibles (représentation multiplicative en tableau, représentation hiérarchisée en arbre..) pour choisir celle des structures conditionnelles "syntaxiques" qui sera le mieux adaptée (traitement par cas , traitements alternatifs emboités).

Il faut insister sur le fait qu'il ne s'agit pas d'acquisitions de méthodes et de contenus qui seraient indépendantes; en effet un rôle central de la connaissance en informatique est de résoudre des problèmes, donc à la fois de modéliser les situations-problèmes et de les traiter de manière opératoire.

Néanmoins la place et le rôle des méthodes dans l'enseignement d'une part, dans les acquisitions par les élèves d'autre part, est un objet de débats. Peu de travaux expérimentaux existent à l'heure actuelle; toutefois les recherches de Hoc ont montré la dépendance entre les types de problèmes traités et l'utilisation des méthodes enseignées dans un cadre professionnel.

On peut dégager trois "pôles" pour des hypothèses sur les modes d'acquisition de méthodes:

a) on peut expliciter a priori les éléments d'une méthode; la mise en oeuvre sytématique de ces éléments dans les problèmes permet à l'élève une acquisition simultanée des contenus spécifiques et de la méthode de traitement; en aidant à la résolution des problèmes la méthode contribue par ailleurs à l'acquisition des contenus eux-mêmes;

b) à partir de situations traitées méthodiquement par l'enseignant, l'élève s'imprègne en quelque sorte du fonctionnement de la méthode sous-jacente et "finit par acquérir la méthode de

travail" (de l'enseignant) (sans qu'un travail explicite lui soit demandé de ce point de vue, ni même que la méthode soit explicitée);

c) après l'acquisition d'un certain nombre de notions de base (dont le contenu et l'ampleur sont à préciser) et de résolution de problèmes, l'élève peut assimiler les objectifs et le contenu d'une méthode générale (présentée alors par l'enseignant) pour aborder ce type de problèmes.

Des travaux en cours, il semble se dégager l'idée que la réponse dépend du type d'élèves auxquels on s'adresse, du type de situations qu'on peut leur faire aborder en relation avec les caractéristiques de la méthode.

Par ailleurs, l'analyse de langages de programmation assez "contrastés" (LSE et PROLOG par exemple) conduit à l'hypothèse qu'en dessous d'un très haut niveau théorique les méthodes de programmation peuvent difficilement être considérées comme indépendantes des types de langages.

La réponse à la question titre de ce paragraphe nous parait donc être d'abord un changement de problématique : quelles situations sont pertinentes pour enseigner des méthodes ; quel moment est-il le plus opportun par rapport à la familiarité des élèves avec le nouveau domaine ; à partir de quelles acquisitions de notions opératoires (qui se réalisent effectivement, avec un langage donné) sera-t-il efficace d'enseigner explicitement une méthode de programmation?

Une réponse appelle des recherches sur l'enseignement de méthodes et sur l'évaluation de ces enseignements, ce dernier point apparaissant un problème très complexe. En effet, l'évaluation de l'enseignement de méthodes ne peut être dissociée de celle de leur efficacité pour résoudre les problèmes, donc aussi de celle de l'acquisition de contenus.

Des travaux en cours dans le domaine de la géométrie, il semble ressortir que l'enseignement de méthodes de résolution modifierait peut-être aussi les représentations des élèves sur le domaine d'enseignement, point dont l'évaluation ne fait guère partie des pratiques courantes du système éducatif.

L'analyse d'un enseignement explicite d'une méthode de traitement de situations complexes (dans un domaine profesionnel) montre qu'on peut enseigner des méthodes à partir de leur explicitation initiale, en tout cas à des sujets ayant déjà des connaissances professionnelles du domaine, et/ou dissociant dans

leurs représentations et leur discours le niveau de la connaissance et celui du savoir sur la connaissance.

Dans l'alphabétisation informatique à des élèves "tout-venant" du second cycle des lycées il ne semble pas que l'on soit dans cette dernière situation; des observations empiriques semblent indiquer que le travail de mise en oeuvre de la méthode reste longtemps dévolu au seul enseignant, et qu'en définitive l'analyse (qui est le plus souvent l'expression de l'enseignement d'une méthode en informatique) est peu assimilée et appropriée par les élèves qui y voient, semble-t-il, surtout un élément du contrat didactique plutôt qu'une utilité pour traiter des problèmes en informatique.

Les travaux en cours conduisent également à l'hypothèse que l'efficacité de l'enseignement de méthode(s) est lié à un certain niveau de complexité des problèmes à résoudre, complexité qui s'accomode peut-être mal d'une ignorance complète de la part des élèves des outils qui seront à leur disposition.

Niveaux de représentation des élèves et niveaux des concepts enseignés

Les recherches dont on dispose aujourd'hui donnent des éléments sur les représentations des débutants et celles de programmeurs professionnels. Les niveaux de représentation des notions et des systèmes sont à l'évidence très différents, et s'accompagnent de niveaux de représentation différents quant aux traitements dont sont justiciables les problèmes proposés.

La question de savoir à quel niveau d'invariants conceptuels on peut initier une formation en informatique selon les acquis antérieurs des élèves (ou étudiants) est une question ouverte tant sur le plan théorique que sur le plan empirique. Le mouvement actuel parmi les informaticiens est de définir pour la programmation un niveau conceptuel élevé. Un objectif est de pouvoir exprimer les concepts et les méthodes indépendamment des langages et des environnements informatiques existants. À cette orientation correspondent des choix didactiques qui mettent en avant l'acquisition de méthodes de programmation, de concepts de haut niveau, la réalisation effective venant en un second temps.

Une évolution possible de cette orientation transposée dans l'enseignement général est de voir se reproduire en informatique l'expérience "bourbakisante" de l'enseignement des mathématiques

dites "modernes" destinée à répondre à des objectifs du même ordre que ceux qui précèdent, face aux évolutions des mathématiques; expérience dans laquelle la forme de l'enseignement nous semble avoir été mal subordonnée au contenu visé pour la transformation de l'enseignement.

Pour notre part nous proposons pour l'alphabétisation une approche définie à partir des considération suivantes. Si on veut faire acquérir des notions en informatique qui restent valides lors d'évolutions importantes des conditions d'interaction homme-machine (langages moins contraignants, aides à la programmation, dépassement de la linéarité de l'éxécution : parallèlisme..) il faut être attentif à ce que les représentations que se construisent les élèves ne deviennent pas des obstacles à des transferts et à des changements de niveau. Toutefois cela n'implique pas ipso facto qu'il faille enseigner à des niveaux différents, mais qu'il faut enseigner en tenant compte de ces niveaux ultérieurs. Il faut donc analyser précisèment pour les notions et les opérations en jeu dans les langages et les environnements actuels ce qui s'intègre dans des invariants de niveau supérieur, et d'autre part analyser les spécificités locales (d'un langage, d'un environnement, d'un dispositif informatique) qui devront être dépassées.

Une telle analyse n'est pas nécessaire seulement pour le travail informatique avec des langages de programmation ; elle l'est également pour l'utilisation de logiciels basés sur l'interactivité et concernant des domaines de problèmes relativement spécifiques (tableurs, gestionnaires de données, traitements de textes, outils graphiques..). Le fait que les interactions soient différentes en PROLOG, LSE ou MULTIPLAN appelle justement à une recherche sur leurs spécificités et sur les invariants qu'on peut dégager dans leur utilisation pour traiter des problèmes.

Par ailleurs si on veut construire une connaissance sur les acquisitions conceptuelles en informatique qui ait un domaine de validité suffisant et garde du sens avec les évolutions de l'informatique comme domaine scientifique, il nous parait nécessaire de s'appuyer sur une analyse épistémologique approfondie pour rechercher dans les activités cognitives des élèves ce qui peut être analysé dans les termes d'un niveau conceptuel assez élevé pour que les résultats observés soient (au moins partiellement) transférables pour l'étude d'acquisitions dans un futur à moyen terme (une ou deux décennies).

La question du niveau auquel on va placer les problèmes proposés aux élèves nous semble un des choix décisifs de l'enseignement, en ce qui concerne les relations entre méthodes, concepts et réalisation effective sur un dispositif informatique.

Notre position est donc que le changement de niveau doit d'abord être dévolu au formateur et au chercheur, pour analyser et construire chez l'élève (ou le sujet en formation) une connaissance qui d'une part soit significative, s'appuye sur les précurseurs disponibles dans d'autres champs conceptuels et d'autres pratiques opératoires, et qui d'autre part puisse être dépassée par la construction de représentations et d'invariants plus généraux (de niveau plus élevé) au cours de la formation ultérieure ou dans la profession.

Evaluation des acquisitions

La question des évaluations est à la fois centrale et complexe. Centrale car c'est un moyen , sinon LE moyen, pour l'enseignant d'apprécier les effets de son enseignement et de faire des choix didactiques sur le long terme ou au cours d'une séquence. Complexe car dès qu'il s'agit de connaissances complexes les acquisitions se situent dans la longue durée, et dans la structuration du domaine des acquis. Cela est vrai pour l'acquisition des notions (concepts et opérations) c'est encore plus vrai pour celle des méthodes qui peuvent modifier les représentations du contenu enseigné et agir sur les acquisitions ultérieures, peut-être encore davantage que sur celles dont leur enseignement est contemporain.

Dans le domaine de l'informatique où les interactions entre contenus et méthodes interfèrent également avec les représentations du dispositif informatique, les évaluations doivent prendre en compte de multiples caractèristiques.

Schématiquement, si on se place comme nous le faisons dans une perspective d'épistémologie génétique, on peut distinguer quatre dimensions dans l'évaluation des acquisitions (en ce qui concerne le contenu): 1) l'extension des notions et des relations internotionnelles (enrichissement du champ conceptuel); 2) l'extension du domaine de validité des acquisitions conceptuelles ou méthodologiques; 3) le changement de niveau des représentations et des traitements; 4) l'évolution des modes de gestion et de contrôle de la connaissance.

Nous allons brièvement donner un exemple pour chacune des trois premières dimensions.

-les acquisitions sur la variable informatique

*peuvent concerner les opérations que l'élève maîtrise la concernant: affectation, test, intervention comme paramètre dans une procédure, variable dans une structure itérative ou récursive : c'est la première dimension de l'analyse ;

*elles peuvent concerner le type de variable auxquelles s'appliquent les opérations, et la richesse de structuration des données qui leur sont liées, ex: utilisation de tests non seulement sur les variables numériques mais sur les chaines de caractères , utilisation de variables booléennes dans les tests et comme valeur de fonctions : seconde dimension ;

*les acquisitions peuvent enfin concerner la capacité à traiter comme des variables des objets de niveau différent, par exemple des relations logiques, ou des procédures , passées en paramètres : troisième dimension.

Un certain nombre de travaux de recherches en psychologie de la programmation ou en didactique de l'informatique donnent des éléments sur ces acquisitions, mais le rapport entre les acquisitions et les situations d'enseignement est encore mal connu et l'effet de l'enseignement de méthodes peu étudié. Le domaine de validité des résultats observés sur des élèves ou sur des adultes (en situation scolaire ou en situation expérimentale) est actuellement un problème ouvert, même si les convergences de résultats indiquent l'existence de phénomènes relativement stables, dans le cadre des modes d'enseignement dominants.

Conclusion

Nous avons essayé de verser au débat quelques éléments sur les analyses des acquisitions cognitives en informatique, et plus précisèment en programmation. Nous avons fait l'hypothèse que l'essentiel des acquisitions en informatique de sujets "tout-venant" se faisait par un processus d'assimilation et d'accomodation, s'appuyant sur l'existence de précurseurs. Nombre de ceux-ci se trouvent dans le domaine mathématique; ils jouent à la fois un rôle producteur de sens et un rôle réducteur dans la mesure où les chmaps conceptuels des mathématiques et de l'informatique sont différents, dans la mesure aussi où existe en informatique un **dispositif** sur lequel les sujets doivent se

construire des représentations.

Les spécificités des acquisitions en informatique tiennent à la fois aux spécificités du contenu et aux spécificités du point de vue des sujets, tout particulièrement à l'existence de ce dispositif informatique et d'une exécution effective des productions. La coordination dans la programmation des points de vue dynamique (en termes d'exécution) et statique (en termes d'états ou de situations) apparait à la fois comme une nécessité et comme une difficulté propre aux acquisitions de savoirs et savoir-faire en informatique.

Une question largement ouverte est celle de l'enseignement de méthodes d'une part pour la modélisation et d'autre part pour l'écriture de programmes. Peu de recherches ont été développées sur ce point pour le domaine propre de l'informatique; celles qui sont engagées (Robert, Rogalski, Samurçay, 1987) indiquent qu'on peut effectivement enseigner des méthodes sur la base d'un certain niveau de connaissances préalables, sans toutefois éliminer l'hypothèse de l'efficacité d'un enseignement initial.

Le niveau des élèves auquel nous nous sommes essentiellement référée est celui d'élèves de second cycle des lycées, initiés à l'informatique, dans un cadre institutionnel. Cela concerne ce que nous appelons l'alphabétisation informatique. Toutefois les questions qui se posent ne nous semblent pas spécifique à cette phase, ni à cette catégorie de sujets.

S'agissant d'un domaine de connaissances complexes, acquises au travers d'un enseignement explicite, notre hypothèse est que la genèse des connaissances peut s'étudier à divers "niveaux" d'enseignement avec les mêmes outils théoriques, et en se posant les mêmes questions. On doit évidemment s'attendre à des différences dans les réponses apportées, car les précurseurs ne sont pas les mêmes, les cadres disponibles non plus, ni les niveaux d'invariants que les sujets peuvent déjà avoir traités dans certains domaines. Ces différences attendues pour des niveaux différents nous semblent également caractériser certains des problèmes de différences individuelles flagrantes mais peu étudiées dans une perspective de genèse des connaissances liée à une problématique didactique.

Nous avons centré cette étude sur les rapports entre les sujets et le savoir, sans développer l'analyse didactique des situations par lesquelles l'enseignant (ou le système d'enseignement) peut gérer la construction de nouvelles

connaissances.

Cela nous parait une base (nécessaire et non suffisante) pour spécifier les objectifs que peut se fixer l'enseignement dans un cadre général donné (en particulier dans le cade de l'option informatique du second cycle des lycées), proposer une organisation possible de séquences d'enseignement, évaluer les acquisitions de savoirs et de savoir-faire.

Toutefois, et nous terminerons sur cette interrogation, une question est ouverte, à propos de l'informatique comme connaissance complexe, avec un historique récent d'un enseignement général : les connaissances contruites chez des élèves sont-elles indépendantes, et dans quelle mesure, des "chemins didactiques" qui ont été choisis pour cette construction? Ce domaine de recherche intégrant intimement les problèmatiques de psychologie cognitive et de didactique ouvre de très larges perspectives à la fois théoriques (sur les invariants de l'acquisition du savoir) et pratiques (sur les effets de choix d'organisation de l'enseignement dans la durée).

ANDERSON J.R., FARELLE R., SAUERS R. (1984) Programming in LISP. Cognitive Science, 8, 87-129.

ARSAC J. (1980) Premières leçons de programmation. Cedic, Paris.

ARSAC J. (1986) L'enseignement de l'informatique dans les lycées. Techniques et Sciences Inforamtiques.

COHORS-FRESENBORG E. (1984) On the representation of algorithmic concepts. Osnabrücker Schriften zue Mathematik. Heft 76.

COHORS-FRESENBORG E. (1986) Empirische Untersuchungen über verschiedene kognitive Struckturen bei algorithmischen Begriffsbildungsprocessen. Actes du colloque franco-allemand de didactique des mathématiques et de l'informatique. CIRM, Marseille-Luminy.

DUBINSKY Ed. (1986) Teaching mathematical induction I Journal of mathematical Behavior (to appear, draft).

DUBINSKY Ed. (1987) Teaching mathematical induction II (draft).

DUCRIM A. (1984) Cours de programmation. 2 vol., Dunod, Paris.

GREEN T.G.R. (1979) The necessity of syntax markers : two experiments with artificial languages. Journal of Verbal Learning and Verbal Behaviour. 18, 401-416.

GREEN T.G.R. (1980) Ifs and Thens : is nesting just for birds ? Software practice and experience. 10, 371-381.

GREEN T.G.R. (1986) Design and use of programming languages. (draft).

HOC J.-M. (1978) Etude de la formation à une méthode de programmation informatique. Le travail humain. 41, 111-126.

HOC J.-M. (1983) Planning and direction of problem-solving in structured programming : an empirical comparison between two methods. Int. J. Man-Machine Studies. 15, 363-383.

LABORDE C, BALACHEFF N., MAJIAS B. (1985) Genèse des concepts d'itération: une approche expérimentale. Enfance. 2-3, 223-239.

MEYER B., BAUDOIN C. (1978) Méthodes de programmation. Eyrolles, Paris.

ROBERT A., ROGALSKI J., SAMURCAY R. (1987) Enseigner des méthodes. Cahiers de didactique IREM Paris-Sud. Université Paris VII, Paris.

ROGALSKI J. (1985) Alphabétisation informatique. Bulletin APMEP. 347, 61-74.

ROGALSKI J. (1986) Les représentations mentales su dispositif informatique. Actes du colloque franco-allemand de didactique des mathématiques et de l'informatique. CIRM, Marseille-Luminy.

ROGALSKI J. (1987) Epistémologie génétique et didactique : pour une théorie de l'acquisition des connaissances complexes. Actes

du colloque SFP : "Les apprentissages : perspectives actuelles". Université de Saint-Denis.

ROGALSKI J., Hé Y. (1987) Logic and mental representations of the informatical device in acquisition of conditional structures by 15-16 years old students. Draft.

ROGALSKI J., SAMURCAY R. (1986) Les problèmes cognitifs rencontrès par des élèves de l'enseignement secondaire dans l'enseignement de l'informatique. Journal Européen de Psychologie de l'Education. 1,2, 97-110.

ROUCHIER A. (1986) Learning recursive calls in building up LOGO procedures. Proceedings of the 10th international conference PME. Londres, 283-288.

ROUCHIER A. (1986) Représentation et mise en scène d'objets informatiques pour l'enseignement. Actes du colloque franco-allemand de didactique des mathématiques et de l'informatique. CIRM, Marseille-Luminy.

SAMURCAY R. (1985) Signification et fonctionnement du concept de variable informatique chez des élèves débutants. Educational Studies inMathematics. 16, 143-161.

SAMURCAY R. (1985) Learning programming : an analysis of looping strategies used by beginning students. For the learning of mathematics. 5,1, 37-43.

SAMURCAY R. (1986) Modèles cognitifs dans l'acquisition des concepts informatiques. Actes du colloque franco-allemand de didactique des mathématiques et de l'informatique. CIRM, Marseille-Luminy.

SAMURCAY R., ROUCHIER A. (1985) De faire à faire-faire : planification de l'action dans une situation de programmation. Enfance. 2-3, 241-254.

SAMURCAY R;, ROUCHIER A. (1987) L'acquisition de la récursivité comme modèle de la programmation itérative. Actes du colloque SFP : "Les apprentissages : perspectives actuelles". Université Saint-Denis.

SAMURCAY R., ROUCHIER A. (1987) Aprentissage de la programmation au collège : un élargissement du champ de focntionnement du schéma récursif. (draft).

SOLOWAY E., EHRLICH K., BONAR J., GREENSPAN J. (1982) What do novices know about programming. in A. Badre & B. Shneiderman (Eds.) Direction in Human-Computer Interaction. Norwood, N.Y., Ablex.

SOLOWAY E., BONAR J., EHRLICH K. (1983) Cognitive strategies and looping constructs : an empirical study. Communication of A.C.M.

SPOHRER J.-C;, SOLOWAY E., POPE E. (1985) A goal /plan analysis of

buggy PASCAL programs. Human-Computer Interaction. 1, 163-207.

van der VEER G., van BEEK J., GRUTS G.A.M. (1986) Learning structures diagrams. Effects of mathematical background, instruction and problem semantics. Colloque : visual aids in programming. Passau.

van der VEER G., TAUBER M.J., WAERN Y., van MUYLIJK (1985) On the interaction between system and user characteristics. Behaviour and information technology. 4, 4, 289-308.

van der VEER G., van der WOLDE J. (1982) Psychological aspects of problem-solving with the help of computer languages. Computer education. 6, 229-234.

WIEDENBECK S. (1985) Novice/expert differences in programming skills. Int. J. Man-Machine Studies. 23, 383-390.