Didactique de l'informatique et acquisitions cognitives en programmation

In Psychologie Française
avec Janine Rogalski

N°32
1987, pp.267-274

Lien internet permanent pour l'article :
https://www.gerard-vergnaud.org/GVergnaud_1987_Acquisitions-Cognitives-Programmation_Psychologie-Francaise-32

LES LANGAGES INFORMATIQUES DANS L'ENSEIGNEMENT

Jean-Michel Hoc
et
Patrick Mendelsohn

LE FRANÇAIS

Décembre 1987 - Tome 32 - 4 - ISSN 0033-2984 - SP - 85 F

3.1.

DIDACTIQUE DE L'INFORMATIQUE ET ACQUISITIONS COGNITIVES EN PROGRAMMATION

Janine ROGALSKI

Chargée de recherche au CNRS.

Principaux thèmes de recherche et d'intérêt : étude des processus d'acquisitions conceptuelles complexes ; résolution de problème en interaction avec l'informatique comme outil et comme objet de connaissance ; acquisition et didactique de méthodes de résolution.

Gérard VERGNAUD

Directeur de recherche au CNRS.

Principaux thèmes de recherche et d'intérêt : psychologie et didactique des mathématiques, langage et pensée, représentations symboliques.

Ad. : Université de Paris V, Laboratoire de Psychologie du développement et de l'éducation de l'enfant, 46, rue Saint-Jacques, 75005 Paris.

SUMMARY

The learning of programming is a complex cognitive activity. The study of this activity requires a complex and theoretical framework : the paper refers to « didactic situations » (situations intented to teach and to face students with new concepts and procedures), to the « conceptual field » on which these concepts and procedures rely, and to important phenomena such as the « didactic transposition » (transformation of knowledge for the purpose of, and through teaching), and the « didactic contract » (teacher's and student's expectations).
Some specificities of computer science are presented concerning on the one hand the process of modeling that is necessarily involved in programming, and on the other hand the characteristics of the physical device used. The primitive ideas in which programming can be rooted, necessarily involve situations outside the field of computers. Invariants of different levels must be recognized by students when they learn programming. The exemple of the concept of variable is given to illustrate different levels of understanding this concept. The learning of control structures is linked to the previous mental representations, and to the methods used in teaching. The case of the iterative structures is developped as an exemple. Finally two models for programming seem to complement each other : a functional model, aimed at finding the informatical solution to a problem, and a practical and effective model, aimed at describing the actual operations to be performed.

INTRODUCTION

Pf N°32-4 Déc. 1987

Les recherches sur la didactique et les acquisitions en informatique ont un double intérêt scientifique et social. D'une part, elles mettent à l'épreuve des concepts théoriques élaborés pour l'étude d'autres acquisitions cognitives complexes et elles ouvrent un large champ d'investigation sur les processus cognitifs de haut niveau. D'autre part, l'imprégnation croissante de la vie sociale par l'informatique pose le problème de la maîtrise, individuelle et professionnelle, de ce phénomène. Ces questions ne concernent pas seulement les activités de programmation, mais, également, et plus généralement, l'utilisation de l'informatique comme outil pour gérer des situations diverses et pour résoudre des problèmes. Elles relèvent d'une double problématique : celle de l'informatique-outil (l'ordinateur comme outil de travail) et celle de l'informatique-objet (l'ordinateur comme objet d'étude).

Nous centrerons plus spécifiquement cette analyse sur l'utilisation de langages de programmation, tout en soulignant que les frontières entre ces langages et les progiciels comme les tableurs, les traitements de texte, etc., s'estompent avec l'évolution actuelle des uns et des autres. Leur place respective dans

→

le système d'enseignement et de formation reste toutefois différente. Les langages de programmation non spécialisés permettent d'aborder l'étude de l'informatique-outil en évitant de restreindre trop le contenu sémantique des problèmes ; ils impliquent peut-être une moindre rupture pour une approche ultérieure de l'informatique-objet. Par ailleurs, les recherches sur des professionnels, en service ou en formation, fournissent des points d'ancrage méthodologiques, et des éléments partiellement transposables dans l'enseignement général.

Nous allons d'abord dessiner le cadre théorique qui nous semble approprié à l'étude des acquisitions cognitives complexes, faisant l'objet d'enseignement. Nous présenterons ensuite les spécificités de l'informatique par rapport à d'autres domaines de savoir (les mathématiques en particulier). Nous aborderons enfin quelques questions qui nous paraissent centrales pour l'étude du processus d'« alphabétisation » informatique.

ACQUISITIONS COGNITIVES COMPLEXES ET DIDACTIQUE

Qu'il s'agisse de savoirs ou de savoir-faire, la connaissance se construit et s'éprouve à travers les problèmes qui peuvent être résolus et les situations que le sujet peut maîtriser. Cette conception interactive des acquisitions cognitives (Vergnaud, 1982) intègre les éléments centraux de l'épistémologie génétique de Piaget. Dans le cas des apprentissages scolaires, cette construction des connaissances s'inscrit dans un processus où les *situations didactiques* (Brousseau, 1980 ; 1981) et les activités cognitives développées par l'élève contribuent de manière décisive à forger ses conceptions, ses représentations et ses compétences. L'analyse de ces situations, de leur organisation conceptuelle et temporelle, apparaît ainsi comme une condition pour étudier acquisitions et processus cognitifs. Pour répondre à la question « quels concepts cet élève a-t-il acquis ? » il est d'abord nécessaire de répondre à la question : « quels problèmes tel concept permet-il de résoudre ? ».

De fait, les analyses épistémologiques mettent en évidence qu'il n'y a pas de relation univoque entre un concept et un problème : un concept est à l'œuvre dans une variété de situations, et dans une situation donnée plusieurs concepts sont en jeu. Ainsi, en informatique la notion de variable peut être liée à de multiples opérations apparaissant dans des situations diverses : affectation, lecture, écriture, tests, procédures, fonctions. Dans un problème exigeant une itération, les fonctions d'initialisation, de test, de mise à jour vont toutes intervenir à propos d'une ou, le plus souvent, de plusieurs variables.

Le cadre théorique des *« champs conceptuels »* a été introduit à la fois pour analyser les situations-problèmes proposées aux élèves, pour interpréter leurs activités cognitives et pour décrire leurs acquisitions (Vergnaud, 1982). Il faut souligner qu'un champ conceptuel n'est pas une donnée directement issue d'un « savoir savant » ou d'une pratique professionnelle : il résulte de la sélection d'un ensemble de concepts, de relations entre concepts et de représentations symboliques, sélection qui implique au préalable le choix d'un niveau d'analyse (Rogalski, 1987). On ne découpe pas les choses de la même manière pour étudier les acquisitions d'un élève de CM2 en LOGO, et celles d'un étudiant scientifique en programmation, même s'il débute en informatique.

Une autre raison du caractère relatif des champs conceptuels réside dans le système d'enseignement lui-même, qui transforme le « savoir savant » et les pratiques professionnelles pour les constituer en objets d'enseignement. La *transposition didactique* (Chevallard, 1985 ; 1987), est partie intégrante des faits d'enseignement, quel qu'en soit le niveau.

L'organisation dans le temps d'une succession de situations didactiques, destinées à confronter l'élève à des problèmes significatifs pour lui et producteurs de nouveaux concepts (par exemple à cause des limites de ses connaissances antérieures), est déjà une transformation du savoir. En effet celui-ci n'est plus organisé par le temps ni par le savoir scientifique seul. Dans un domaine aussi jeune que l'informatique qui n'a guère de tradition d'enseignement, la variété des transpositions didactiques est considérable, en particulier dans les cours d'initiation. Soulignons que cette variété des transpositions didactiques pose un problème redoutable pour la recherche de phénomènes invariants au cours des acquisitions et pour la recherche d'obstacles épistémologiques éventuels.

Un autre élément important du processus d'enseignement, qui modifie le cours des acquisitions, est l'existence, entre les élèves et l'enseignant d'un *contrat didactique :* ce contrat implicite fixe les enjeux des situations didactiques et les attentes réciproques de l'enseignant et des élèves quant au contenu de leur activité. Par exemple, une méthode de programmation peut exiger une certaine forme de représentation de la solution (graphique par exemple), préalable à l'écriture du programme. La fonction de cette forme de représentation est de mettre en évidence des éléments-clés de l'organisation de la solution, pour structurer le programme, en vue d'assurer sa validité et sa lisibilité. Faute de disposer des situations didactiques qui donneraient du sens à cette fonction, l'enseignant peut imposer le contrat que la représentation attendue doit accompagner l'écriture du programme, même si cette représentation a été constituée après coup. Il semble raisonnable de rechercher du côté des effets de contrat didactique la raison de certains échecs notamment lors des phases d'initiation.

L'étude des acquisitions en informatique en général, et en programmation en particulier, demande que soit pris en compte et analysé le processus

d'enseignement dans lequel elles se situent. Les concepts théoriques présentés ci-dessus visent justement à permettre l'analyse de ce processus.

LES SPÉCIFICITÉS DE L'INFORMATIQUE

Fondamentalement, et plus encore que les mathématiques, l'informatique est une *discipline de service*. Les concepts qui lui sont propres en tant que discipline scientifique autonome servent comme « outils » de modélisation et de résolution de problèmes posés dans d'autres domaines de savoirs ou de pratiques.

Les acquisitions en programmation comportent donc nécessairement, d'une part des *connaissances conceptuelles*, sur l'informatique, d'autre part des connaissances sur les *processus de modélisation* et sur les méthodes de programmation (comment passer du problème au programme). Il faut y ajouter les connaissances relatives à l'exigence de réalisation effective : les solutions doivent être non seulement calculables mais exécutables en un temps limité et fournies sous une forme appropriée à l'utilisateur humain.

Si la modélisation peut être un objectif essentiel des mathématiques, la programmation n'est pas réductible pour autant à l'activité logico-mathématique sur laquelle elle s'appuie : l'algorithmique. L'existence de concepts et de méthodes spécifiques tient aux interactions qui existent entre la représentation des données, le traitement logique, le langage, les techniques : elles conduisent à des formes propres de modélisation. Par exemple, la notion de variable informatique intervient, dans toute une classe de langages, en liaison avec l'instruction d'affectation. Cette instruction ne s'identifie pas à l'égalité définitionnelle des mathématiques ; dans les structures itératives son statut est en fait celui d'une fonction du temps de l'exécution. Même dans un langage comme PROLOG, étroitement lié à la logique dans ses concepts et sa syntaxe, il existe des spécificités liées à l'exécution du programme qui conduisent par exemple la conjonction de propositions à ne pas toujours être commutative, au sens suivant : le texte du programme est écrit en termes identiques à ceux de la conjonction de propositions, mais cette écriture n'est pas nécessairement commutative pour l'exécution.

L'activité de modélisation implique la maîtrise de concepts informatiques mais ne s'y réduit pas. L'expérience du travail de professionnels a montré qu'il s'agissait d'une activité complexe dont la gestion devenait souvent peu efficace pour des problèmes d'une certaine ampleur. Il est apparu ainsi la nécessité d'une formation spécifique à des *méthodes de programmation*. Les travaux de Hoc (Hoc, 1978 ; 1983) ont mis en évidence le rôle des conditions de mise en œuvre et l'existence d'interactions avec la nature des problèmes. Les recherches sur les processus d'enseignement (et d'acquisition) de méthodes restent pour l'essentiel à développer (Robert, Rogalski, Samurçay, 1987).

Par ailleurs, toute résolution de problème par l'informatique exige la prise en compte de l'existence du *dispositif informatique (DI)* : ordinateur avec ses périphériques, ses interfaces logicielles et matérielles, le langage de programmation utilisé. Les sujets doivent donc interpréter la production effective d'une solution non seulement du point de vue de sa logique interne mais aussi du dispositif informatique dans lequel cette solution sera réalisée (Rogalski, 1986). On observe ainsi un phénomène de « distanciation » analogue à celui introduit dans les domaines de la physique ou de la technologie par l'usage d'un instrument complexe. Les problèmes soulevés par l'apprentissage de règles de fonctionnement et d'expression des procédures de traitement (voir les 1re et 2e parties de ce volume) relèvent en partie de ce rôle spécifique du DI.

Enfin, l'informatique présente une autre spécificité qui pèse sur le processus d'apprentissage et d'enseignement. Il n'existe pas pour l'informatique un arrière-plan d'activités intellectuelles « spontanées », qui pourraient être développées par l'enfant indépendamment de toute volonté didactique extérieure, comme c'est le cas pour les premières notions numériques et spatiales, par exemple. L'ordinateur n'est pas pour l'instant un objet sur lequel l'enfant exercerait librement des activités explorant le « monde » informatique et se poserait des problèmes le conduisant à élaborer des notions informatiques. En tout état de cause l'informatique – comme d'autres domaines technologiques complexes – contient un savoir humain considérable qui s'exprime dans les fonctions remplies et dans les logiciels. Cela donne à l'ordinateur des propriétés d'interaction tout à fait spécifiques. Une des conséquences est que les *précurseurs* susceptibles de donner du sens aux concepts informatiques (rôle *producteur*) doivent être recherchés hors du domaine informatique. Une analyse comparative de leur fonctionnement dans leur domaine d'origine et dans le nouveau champ conceptuel de l'informatique est nécessaire pour comprendre et circonscrire le rôle *réducteur* de ces précurseurs : l'élève transpose des caractères non adéquats des précurseurs, et ceux-ci peuvent ainsi former des obstacles à la construction des invariants propres au champ informatique.

Par exemple, le concept de variable informatique a comme précurseur possible celui de variable mathématique, mais le caractère d'invariant (statique) que présente la relation fonctionnelle en mathématiques peut constituer un obstacle à la représentation de la modification (dynamique) de la valeur d'une variable informatique au cours de l'exécution du programme. Les procédures de résolution familières, « à la main », peuvent, dans une certaine approche, jouer un rôle producteur de sens mais elles peuvent aussi jouer un rôle réducteur insidieux par tout l'implicite qu'elles contiennent pour le sujet.

HIÉRARCHIE D'INVARIANTS CONCEPTUELS : L'EXEMPLE DES VARIABLES

Nous voudrions montrer sur l'exemple des variables les différents niveaux auxquels différents *invariants conceptuels* peuvent être construits dans le domaine de la programmation. Le concept de variable informatique nécessite en premier lieu que soit reconnue l'invariance du nom de la variable tout au long du programme, quelles que soient les valeurs prises par cette variable (s'il y a des calculs conditionnels par exemple) ; cet *invariant* correspond à la fois à la *désignation* de la variable dans le texte du programme, et à la *référence* en mémoire dans le DI. La possibilité de choisir un nom significatif (« longueur » par exemple pour une figure en LOGO) facilite la construction de cet invariant « en acte », mais la propriété de biunivocité n'est pas nécessairement assurée : par exemple le même nom de variable peut être utilisé pour désigner dans un programme la longueur du côté d'un carré et celle d'un triangle. La plupart des méthodes de programmation donnent un statut explicite à cet invariant et demandent le listage des noms utilisés pour les « objets » du programme avec leur *rôle fonctionnel dans le programme*. Telle variable, désignée par son nom, devient ainsi dans cette liste un « objet » parmi d'autres, ce qui suppose qu'une catégorie de niveau supérieur soit appréhendée : sous des noms différents on désigne des variables différentes, qui n'en sont pas moins toutes des variables, et dont le rôle différencié dans le programme peut être analysé et explicité.

Les situations didactiques qui donnent du sens à ce changement de niveau mettent en scène des problèmes dans lesquels la définition informatique de la variable ne peut se réduire à une instruction, mais en implique nécessairement plusieurs. En effet ces problèmes comportent des objets dont la définition est soit conditionnelle, soit itérative, soit récursive.

Un nouvel invariant conceptuel apparaît quand on considère les situations itératives : dans une boucle il faut considérer la variable comme fonction (au sens mathématique du terme). Ainsi dans le fragment de programme qui affiche la suite des nombres jusqu'à 100 :

```
n : = 0
TANT QUE n ≠ 100 FAIRE
n : = n + 1
AFFICHER (n)
FIN TQ
```

la variable n représente et désigne la suite des nombres, engendrée par itération, et non pas seulement un nombre qui aurait un rôle particulier dans le problème. Ce changement de niveau est particulièrement important et délicat : il correspond à une sorte d'« encapsulation » de notions : les valeurs successives de la même variable au cours de l'exécution sont prises comme un tout : la relation fonctionnelle, et c'est sur cette fonction qu'il faut raisonner. Ajoutons que la fonction dépend de l'étape dans la boucle, élément implicite du texte du programme. De plus cet invariant n'est pas isolé : c'est un élément de l'*invariant de boucle* c'est-à-dire de la *relation entre variables-fonctions* qui est conservée tout au long de l'exécution de la boucle. Cet invariant de haut niveau peut être construit « en acte », comme expression de la transformation de la variable dans le corps de boucle (*invariant dynamique* lié à l'exécution). Mais la construction de la relation fonctionnelle comme telle (invariant statique, indépendant du déroulement temporel du programme) exige une intervention didactique sur la signification de cet invariant.

Prenons comme exemple le fragment de programme qui calcule et affiche la somme des 100 premiers nombres entiers :

```
nombre : = 0
somme : = 0
TANT QUE nombre < 100 FAIRE
    nombre : = nombre + 1
    somme : = somme + nombre
FIN TQ
AFFICHER (somme)
```

Deux variables-fonctions sont impliquées : la suite « nombre » des nombres à additionner et la fonction « somme ». L'observation d'élèves de seconde (Rouchier, Samurçay et al., 1984) a montré que les invariants « dynamiques » correspondant aux transformations à effectuer dans le corps de boucle n'étaient pas de difficulté analogue. Alors que la suite des nombres a été traitée pratiquement comme un compteur, avec le rôle fonctionnel « augmenter de 1 », il a été difficile pour les élèves de dégager le fait qu'à chaque étape de calcul ils ajoutaient à la somme déjà obtenue (*état* de la variable « somme ») le nombre suivant. Nous ne connaissons pas de recherche ayant analysé, pour des élèves en initiation, les processus de construction de l'invariant comme relation « statique », dont la difficulté est attestée par les enseignants. Les recherches sur l'enseignement de la récursivité en LOGO apportent des éléments importants, mais elles concernent le champ spécifique de la production d'objets graphiques, et le domaine de validité des observations recueillies reste une question ouverte.

L'emboîtement des invariants liés à la notion de variable informatique montre ainsi que les apprentissages en programmation relèvent de processus complexes, dont l'étude doit porter sur un temps relativement long. Cet exemple montre aussi la nécessité de mener conjointement une analyse épistémologique des notions en jeu et une analyse de la complexité des opérations cognitives qu'elles impliquent. Ce sont là des bases nécessaires pour l'élaboration de situations didactiques, moyen d'expérimentation privilégié sur les processus didactiques.

DIDACTIQUE ET ACQUISITIONS EN PROGRAMMATION

Un programme exprime et calcule une fonction d'un ensemble de données sur un ensemble de résultats :

$$\mathcal{F} : \mathcal{D} \rightarrow \mathcal{R}$$

L'objet même qu'est un programme peut être considéré sous deux points de vue. Le point de vue fonctionnel considère le but, la finalité du programme : c'est un point de vue plutôt « déclaratif » et relativement statique, orienté vers le problème à résoudre, pour lequel le programme propose une solution effectuable. L'autre point de vue considère la réalisation, et s'intéresse aux « calculs » au sens large c'est-à-dire à la suite de changements d'états : c'est un point de vue plutôt procédural et dynamique, orienté vers le texte du programme qui répond au problème. Le passage du problème au programme implique deux types d'activités : comment construire un programme (sur des « objets du monde » ou sur des « objets informatiques ») et comment représenter les objets sur lesquels on travaille ; de fait les deux activités (algorithme et structuration de données) présentent des aspects à la fois procéduraux et déclaratifs (Pair, 1987) : il ne faut donc pas utiliser de manière schématique la distinction introduite entre les deux « modèles ».

Les invariants conceptuels que l'on considère, le découpage d'un champ conceptuel et le choix des situations didactiques peuvent différer de manière importante selon le ou les points de vue choisis : on peut trouver une illustration de cette question à propos des premières acquisitions sur les procédures récursives en LOGO (Samurçay et Rouchier, 1987). Les méthodes de programmation, qui visent à orienter le passage du problème au programme, en appellent fortement au modèle fonctionnel tandis que l'acquisition de représentations adéquates sur la séquentialité d'exécution des programmes ferait plutôt appel au modèle de réalisation.

De manière dominante, les recherches sur les acquisitions en programmation ont mis l'accent sur les problèmes cognitifs soulevés par les procédures de traitement, presque toujours associées à un langage de programmation et/ou un DI déterminés (Anderson et al., 1984 ; Cohors-Fresenborg, 1988 ; Green, 1986 ; Kaune, 1985 ; Laborde et al., 1985 ; Méjias, 1985 ; Mendelsohn, 1985-1986 ; Rogalski et Samurçay, 1986 ; Rouchier, 1986 ; Samurçay, 1986, Samurçay et Rouchier, 1985 ; Soloway et al., 1982 ; cf. aussi la 2e partie de ce volume). Les aides logicielles pour débutants sont également plutôt orientées par la représentation du programme comme organisation de « calculs » (Spohrer, Soloway, Pope, 1985 ; di Sessa, 1985). Il est possible que cela reflète un changement de niveau de complexité quand on passe du *modèle de la réalisation* (orienté vers la description des « calculs » dans le programme) au *modèle fonctionnel* (orienté vers l'analyse de la fonction du programme par rapport au problème).

En tout état de cause, l'essentiel des études concerne l'apprentissage des structures de contrôle : conditionnelles, itératives et récursives. La variable informatique est un élément important du réseau des concepts impliqués dans chacune de ces structures. L'analyse précédemment faite de la hiérarchie des niveaux conceptuels s'intègre donc dans l'étude de leur apprentissage. Par ailleurs, un caractère fondamental des structures de contrôle est la rupture qu'elles organisent avec la linéarité du texte du programme ; on peut donc s'attendre à ce que l'ordre dans le programme des différents éléments constitutifs de ces structures joue un rôle dans la plus ou moins grande difficulté d'appropriation par des élèves.

Les problèmes d'apprentissage des structures conditionnelles et de la récursivité sont abordés ailleurs dans ce volume ; nous rappellerons ici les résultats essentiels sur l'acquisition et l'enseignement de l'itération, dans des langages « impératifs » comme PASCAL, BASIC, ou LSE.

L'itération permet la répétition d'un traitement un nombre quelconque de fois ; ce nombre peut être connu à l'avance ou peut dépendre de l'état obtenu au cours du traitement. La tâche de construction d'une boucle comprend trois éléments :
– la planification répétitive du traitement, c'est-à-dire l'élaboration et l'expression de l'invariant dans les « calculs » à répéter : (corps de boucle).
– l'identification du statut fonctionnel du contrôle d'arrêt : sur quelle variable porte-t-il ? quand intervient-il dans la boucle ? (test) ;
– la détermination de l'état initial dans lequel doivent se trouver les variables de la boucle (initialisation).

Lorsque le test porte sur la valeur d'une variable modifiée dans le corps de boucle, deux ordres sont possibles :
– corps de boucle/test : structure RÉPÈTE... JUSQU'À...
– test/corps de boucle : structure TANT QUE... FAIRE...

Ces deux formes ne sont pas également accessibles, ni dans l'apprentissage, ni dans la mise en œuvre. Les modèles spontanés de traitement répétitif font apparaître l'ordre suivant : description de l'action, identification de la répétition et, enfin, contrôle ; le compteur du nombre d'actions est placé après la description de l'action (Rouchier et al., 1984 : Samurçay et Rouchier, 1985). De plus l'anticipation et l'explicitation de l'arrêt de la répétition est une source de difficulté, car le contrôle est implicite dans le fonctionnement habituel des élèves (Laborde et al., 1985). Les propriétés des précurseurs de traitement « à la main » des répétitions contribuent à expliquer que la maîtrise de la boucle TANT QUE... qui obéit au schéma inverse, et dans laquelle l'action peut ne jamais avoir lieu, pose plus de problèmes (Soloway, Ehrlich, Bonar et Greenspan, 1982).

Par ailleurs, l'élaboration de l'invariant de boucle est une opération difficile. L'élève doit en effet identifier d'une part les variables en jeu et d'autre part la relation entre elles qui doit être conservée tout

au long de la boucle. L'observation d'élèves lors de l'écriture d'un programme itératif (avec la forme RÉPÈTE...) a montré la complexité du processus d'élaboration (Rogalski, 1984 ; 1985). (L'invariant de boucle est ici l'ensemble des instructions qui permet de calculer le résultat : toutefois en tant que relation entre variables, il est par ailleurs lié à l'invariant de boucle au sens informatique, bien qu'il ne soit pas traité comme tel par les élèves).

La représentation de la liste des états successifs des variables, c'est-à-dire des situations successives au cours du déroulement du programme, s'est avérée un élément productif dans cette élaboration. L'appropriation de la structure itérative assurant l'écriture de programmes valides sans les procédures, coûteuses et peu fiables, d'essais et erreurs pour l'initialisation et pour l'arrêt de la boucle, exige peut-être le passage de la représentation en termes d'*actions* (liée au modèle dynamique de la réalisation) à une représentation en termes d'*états* (liée au modèle fonctionnel statique). La construction de situations didactiques permettant ce passage est sans doute un des problèmes clés dans l'enseignement de la programmation.

CONCLUSION

Nous avons montré dans la première partie que l'étude du procès d'enseignement et des processus d'acquisitions en programmation ne pouvait se réduire aux analyses sur le contenu du savoir et les rapports entre le sujet (l'élève) et le savoir. Nous avons souligné en particulier l'existence de deux « points de vue » sur la programmation, que nous avons schématisés comme un modèle de réalisation et un modèle fonctionnel. Les situations didactiques correspondant à chacun de ces points de vue vont différer, et les notions qu'elles visent à construire diffèrent a priori aussi dans la phase d'initiation. Dans un cas, l'accent sera mis d'abord sur la réalisation du programme répondant à des traitements d'objets informatiques, dans l'autre cas, l'accent sera mis sur l'organisation du traitement des objets du problème. Cela nous semble correspondre à la différence entre enseignement par schémas et enseignement par méthodes présentée par Arsac (Arsac, 1986). Dans le cas des schémas « le professeur présente [...] des programmes schématiques représentant les constructions de base de la programmation [...]. Les élèves résolvent d'abord des problèmes simples ne mettant en jeu qu'un seul schéma, puis ils passent à des problèmes plus complexes qu'il faut d'abord décomposer... ». Dans le second cas « [la construction du programme] découle de la suite de situations qu'il doit engendrer. Il faut décrire [...] les situations initiale et finale. [...] Il faut jalonner le chemin par des situations intermédiaires. » La méthode guide le choix des situations intermédiaires et la recherche d'une description efficace des situations.

Les recherches existant sur d'autres contenus indiquent que les situations-problèmes appropriées à l'acquisition de méthodes (et à l'acquisition de contenus à travers et par des méthodes) doivent être suffisamment complexes et permettre l'existence de plusieurs « techniques » de résolution. En revanche, il semble que l'acquisition par schémas soit plutôt liée à l'enrichissement et la coordination de schémas primitifs. Peut-on évaluer avec les mêmes moyens des connaissances initiales produites par des voies si contrastées ?

Nous voudrions, en guise de conclusion, proposer de développer la problématique d'étude de la didactique et des acquisitions en programmation avec la perspective d'analyser et de construire l'établissement (par le sujet) d'une double relation : d'une part relation entre la représentation dynamique d'un programme liée à son exécution sur un dispositif informatique donné, et la représentation statique exprimant des relations entre objets informatiques, d'autre part relation entre un modèle fonctionnel et un modèle de réalisation de l'activité de programmation elle-même.

RÉFÉRENCES BIBLIOGRAPHIQUES

ANDERSON J.R., FARELL R., SAUERS R. – Programming in LISP. *Cognitive Science*, 1984, *8*, 87-129.

ARSAC J. – L'enseignement de l'informatique dans les lycées. *Techniques et Sciences Informatiques*, 1988, *5*, *3*, 230-251.

BROUSSEAU G. – Problèmes de l'enseignement des décimaux. *Recherches en didactique des mathématiques*, 1981, *2*, 37-125.

CHEVALLARD Y. – *La transposition didactique : du savoir savant au savoir enseigné*. Grenoble, La Pensée Sauvage, 1985.

CHEVALLARD Y. – Théorie didactique et étude de l'enseignement de l'algèbre : chronique d'une recherche. *Actes du Colloque du GRECO CNRS : Didactique et acquisition des connaissances scientifiques*, Paris, 1987.

COHORS-FRESENBORG E. – Empirische Untersuchungen über verschiedene kognitive Strukturen bei algorithmischen Begriffsbildungsprozessen. *Actes du Colloque franco-allemand de didactique des mathématiques et de l'informatique*, CIRM, Marseille-Luminy, 1988.

Di SESSA – A principled design for an integrated computational environment. *Human-Computer Interaction*, 1985, *1*, 1-47.

GRAM Anna. – *Raisonner pour programmer*. Paris, Dunod, 1988.

GREEN T.G.R. – Design and use of programming languages, 1988 (draft).

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

HOC J.-M. – Planning and direction of problem-solving in structured programming : an empirical comparison between two methods. *Int. J. Man-Machine Studies*, 1985, *15*, 363-383.

KAUNE C. – *Schüler denken am Computer. Eine Untersuchung über den Einfluss von Repräsentationsformen und kognitiven Strategien beim Konstruieren und Analysieren von Algorithmen*. Schriftreihe des Forschungsinstituts für Mathematikdidaktik, Osnabrück, 1985, *5*.

LABORDE C., BALACHEFF N., MEJIAS B. – *Genèse des concepts d'itération : une approche expérimentale*. Enfance, 1985, *2-3*, 233-239.

MEJIAS B. – *Difficultés conceptuelles dans l'écriture d'algorithmes itératifs chez les élèves de collège.* 1985, Thèse de 3e cycle, USM, Grenoble.

MENDELSOHN P. – Activation de schèmes de programmation et mémorisation de figures géométriques. *Journal Européen de Psychologie de l'Éducation,* 1986, *1-2,* 127-138.

PAIR C. – Programmation, langages et méthodes de programmation. *Travail Humain* (à paraître).

ROBERT A., ROGALSKI J., SAMURÇAY R. – Enseigner des méthodes. *Cahier de didactique,* 1987, *38,* IREM, Université Paris VII.

ROGALSKI J. – Sur une séquence didactique en informatique. *Cahiers du séminaire de didactique des mathématiques et de l'informatique IMAG,* 1984, *90,* IMAG, Grenoble, 1-21.

ROGALSKI J. – Alphabétisation informatique : problèmes conceptuels et didactique. *Bulletin APMEP,* 1985, *347,* 61-74.

ROGALSKI J. – Épistémologie génétique et didactique : pour une théorie de l'acquisition des connaissances complexes. *Actes du colloque SFP : « Les apprentissages : perspectives actuelles »,* Université de Saint-Denis, 1987.

ROGALSKI J. – Acquisition de savoirs et savoir-faire en informatique. *Cahier de didactique des mathématiques,* 1987, *43,* IREM, Université Paris VII.

ROGALSKI J., SAMURÇAY R. – Les problèmes cognitifs rencontrés par des élèves de l'enseignement secondaire dans l'enseignement de l'informatique. *Journal Européen de Psychologie et de l'Éducation,* 1988, *1-2,* 97-110.

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

ROUCHIER A., SAMURÇAY R., ROGALSKI J., VERGNAUD G., DIISPLAND J.-C., LAUBIN J.-M., LANDRE C., SARFATI G., PIGEONNAT J.-F., FERRAND Y. – *Concepts informatiques et programmation. Une première analyse en classe de seconde des lycées,* Rapport de recherche, CNRS, IREM Orléans, 1984.

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

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

SAMURÇAY R., ROUCHIER A. – De faire à faire-faire planification de l'action dans une situation de programmation. *Enfance,* 1985, *2-3,* 241-254.

SAMURÇAY R., ROUCHIER A. – L'acquisition de la récursivité comme modèle de la programmation itérative. *Actes du colloque SFP : « Les apprentissages : perspectives actuelles »,* Université Saint-Denis, 1987.

SOLOWAY E., EHRLICH K., BONAR J., GREENSPAN J. – What do novices know about programming. In A. BADRE & B. SHNEIDERMAN (Eds.). *Direction in Human-Computer Interaction,* Norwood, N.Y., Ablex, 1982, 27-54.

SPOHRER J. G., SOLOWAY E. – Analysing the high frequency bugs in novice programs. In E. SOLOWAY and S. IYENGAR (Eds). *Empirical studies of programmers,* Norwood, N.J. Ablex, 230-251, 1986.

SPOHRER J.-C., SOLOWAY E., POPE E. – A goal/plan analysis of buggy Pascal programs. *Human-Computer Interaction,* 1985, *1,* 163-207.

VERGNAUD G. – Cognitive and developmental psychology and research in mathematics education : some theoretical and methodological issues. *For the learning of mathematics,* 1982, *3, 2,* 31-41.

RÉSUMÉ

L'apprentissage de la programmation est une activité complexe, dont l'étude nécessite un cadre théorique suffisamment large. Cet article fait référence aux « situations didactiques », aux « champs conceptuels » dont font partie les nouveaux concepts et procédures de l'informatique, à la « transposition didactique » et au « contrat didactique » qui règle les attentes réciproques des enseignants et des élèves.

Des spécificités de la programmation sont présentées, liées d'une part au processus de modélisation et d'autre part à l'intervention d'un dispositif informatique physique : elles conduisent à rechercher des précurseurs aux notions informatiques hors de ce domaine. Des invariants de différents niveaux doivent être reconnus par les élèves lorsqu'ils apprennent à programmer : l'exemple du concept de variable informatique est développé pour analyser les différents niveaux de compréhension de ce concept.

Les apprentissages des structures de contrôle de la programmation dépendent des conceptions initiales des élèves et des choix de l'enseignement. Le cas de l'itération illustre les difficultés de ces apprentissages. Deux modèles de la programmation semblent complémentaires : un modèle fonctionnel, orienté vers la recherche d'une solution informatique à un problème, et un modèle de réalisation, orienté vers la description des opérations qui doivent être effectivement réalisées dans le programme.