Enseignement de méthodes de programmation dans l'initiation à l'informatique

Janine Rogalski

ENSEIGNEMENT DE METHODES DE PROGRAMMATION DANS L'INITIATION A L'INFORMATIQUE

Janine ROGALSKI

CNRS, PARIS V, Laboratoire PSYDEE
46, rue Saint-Jacques 75005 Paris

ENSEIGNEMENT DE METHODES DE PROGRAMMATION DANS L'INITIATION A L'INFORMATIQUE

Janine ROGALSKI

CNRS, PARIS V, Laboratoire PSYDEE
46, rue Saint-Jacques 75005 Paris

Mars 1988

L'enseignement de l'informatique à des élèves (ou des étudiants), qui ne sont pas a priori de futurs professionnels de l'informatique, pose des problèmes didactiques aussi épineux que ceux rencontrés par les enseignants des différentes disciplines "traditionnelles". L'informatique y apparaît à la fois comme un objet, relevant d'un domaine scientifique avec ses concepts propres, et comme un outil pour contribuer à résoudre les problèmes d'autres domaines.

Parmi ses spécificités - du point de vue du contenu et du point de vue des élèves[1] - celle d'être une discipline de service, liée à de multiples pratiques professionnelles, a (indirectement ?) conduit au problème de l'enseignement de méthodes de programmation. La réflexion sur l'efficacité du travail des programmeurs professionnels a en effet convergé, vers la fin des années soixante, avec des approches de nature formelle et les travaux sur la programmation structurée (Dahl, Dijktra, Moare, 1972)[2].

Cela a conduit à concevoir la structure d'un programme de façon hiérarchique et modulaire, à élaborer des méthodes de programmation dans le domaine professionnel et à engager une réflexion sur les conséquences à en tirer dans le domaine de l'enseignement de la programmation, indépendamment de la perspective professionnelle.

L'hypothèse didactique était "qu'une programmation méthodique dès les premiers apprentissages évite de prendre des habitudes de programmation empirique et désordonnée, tout en facilitant la résolution des problèmes de programmation" (Rogalski, Samurçay, Hoc).

---

(1) La question des spécificités de l'informatique est liée étroitement au problème des diverses transpositions didactiques qui peuvent être utilisées pour passer d'une connaissance du domaine scientifique ou des pratiques professionnelles à un objet d'enseignement. Les deux types de transposition sont également en relation avec les représentations qu'ont les enseignants sur l'informatique et ce qu'ils souhaitent que se représentent les élèves. Le point de vue des spécificités de l'informatique est développé dans (Rogalski, 1987a).

(2) On peut trouver dans (Rogalski, Samurçay, Hoc) un bref aperçu historique destiné à situer l'apparition et le statut des méthodes par rapport à l'activité du programmeur.

Plusieurs années d'expérience d'enseignement de l'informatique dans l'option des lycées conduisent à penser que les difficultés d'enseignement de méthodes ont été sous-estimés. Avant de pousser à des révisions déchirantes (C. Pair,1988), il n'est pas inutile d'analyser plus précisément le ou les objets d'enseignement relatifs aux méthodes de programmation pour essayer d'identifier les propriétés de situations d'enseignement propres à faire acquérir par les élèves cette approche méthodique reconnue comme nécessaire.

LES MÉTHODES DE PROGRAMMATION COMME MÉTHODES DE RÉSOLUTION DE PROBLÈME

Une méthode peut être considérée comme un outil d'aide à la résolution de problèmes : elle ne décrit pas les procédures à utiliser, mais guide l'utilisateur pour la recherche de stratégies de résolution de problèmes d'une certaine classe. Une méthode exprime un processus commun de recherche de modes de résolution, processus valable pour une classe de situations qui ont un ensemble de points communs.

La figure 1 schématise le lien entre une méthode et une classe de problèmes.

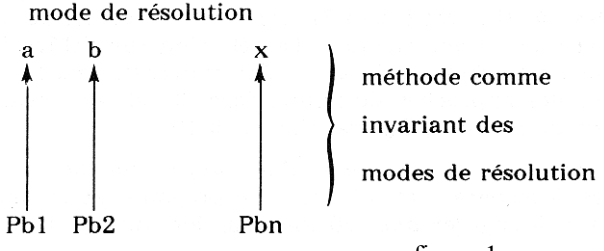

figure 1

Plus la classe de situations que la méthode vise à traiter sera large, plus la méthode sera générale et s'éloignera de l'expression directe en termes de procédures. On peut constater un tel "mouvement" en comparant les approches méthodologiques de A. Ducrin (qui vise la formation à la programmation) et de A. Gram (qui s'adresse à des programmeurs) (A. Ducrin, 1984 ; A. Gram, 1986).

Le plus souvent une méthode est élaborée par une collectivité professionnelle, sur la base d'un ensemble de connaissances collectives. Elle explicite (ou vise à expliciter) ce qu'il y a d'invariant dans des pratiques efficaces de résolution d'un certain type de problèmes ; le caractère prescriptif de la méthode ne reflète pas nécessairement les activités de résolution de problèmes "familiers" mais des aides utiles pour organiser cette activité dans des problèmes "difficiles".

En ce qui concerne les modes d'acquisition de méthodes, on peut faire deux hypothèses contrastées :

1) Une présentation explicite des concepts de la méthode (faite par l'enseignant) suivie d'une mise en oeuvre par les élèves sur des problèmes particuliers - pour lesquels la méthode est fonctionnelle - leur permet de s'approprier la méthode comme outil pour guider une démarche efficace de résolution ;

2) L'appropriation de la méthode est le résultat de l'élaboration par l'élève de procédures pour résoudre des problèmes représentatifs d'une classe de situations ; l'élève lui-même, ou l'enseignant, dégage ensuite la démarche commune qui peut guider la recherche de ces procédures.

L'expérience de l'enseignement en mathématiques, comme celle sur les programmeurs professionnels, conduit à l'hypothèse raisonnable que la seconde voie d'acquisition peut être le fait d'une (petite) minorité de sujets. Des recherches et observations de terrain ont par ailleurs montré la possibilité d'enseigner explicitement des méthodes par la voie 1, à des élèves ou stagiaires ayant déjà un certain nombre de connaissances dans le domaine concerné (Robert, Rogalski, Samurçay, 1987). Ces recherches indiquent par ailleurs qu'une complexité "suffisante" des problèmes traités par les élèves est nécessaire pour que le caractère d'"outil" apporté par la méthode apparaisse. L'approche de tels problèmes complexes paraît plus productive si les élèves sont dans une situation de travail collectif : une telle situation conduit à expliciter les stratégies ou ébauches de stratégies ; par ailleurs, l'existence de connaissances et de points de vue différents permet au groupe d' "entrer" dans un problème complexe moins difficilement qu'à un élève seul.

Ces analyses et résultats généraux conduisent 1) à spécifier pour la programmation la nature des problèmes et des méthodes associées ; 2) à rechercher dans quels cas les méthodes sont centrées sur l'utilisation efficace de connaissances acquises et dans quels cas elles sont liées à l'acquisition même d'un contenu ; 3) à chercher quels types de problèmes et de situations de résolution de problèmes sont adaptés à l'acquisition de telle ou telle méthode ; 4) à étudier quels "moments" au cours de l'enseignement de l'informatique vont être opportuns pour introduire explicitement telle ou telle méthode (question évidemment liée à la question 3).

PROBLÈMES ET MÉTHODES EN PROGRAMMATION

Le premier point à préciser est ce que l'on entend par "résoudre un problème" de programmation. Une définition générale est donnée par C. Pair comme le passage d'un problème sur des "objets du monde" au texte d'un programme exécutable par un dispositif informatique donné.

La figure 2 schématise les deux "dimensions" de représentation des objets et de traitement des données qui interviennent dans la résolution d'un problème de programmation.

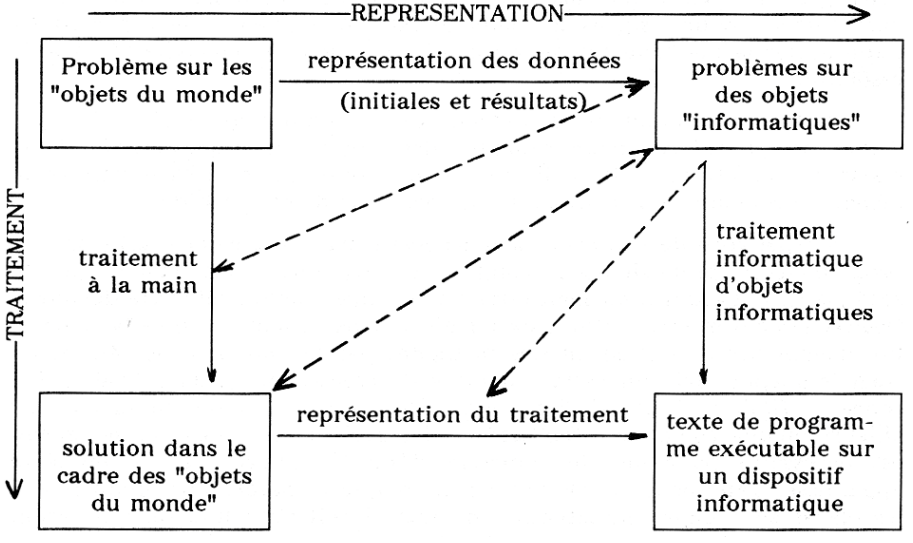

figure 2

(Les flèches en "tirets" représentent des interactions entre les deux voies contrastées de résolution : traitement sur les "objets du monde" puis représentation informatique ou représentation par des "objets informatiques" puis traitement).

Le problème de programmation peut changer de caractéristique selon le domaine concerné, selon la "distance" entre le problème posé dans les objets-du-monde et le texte du programme. En particulier la spécification peut constituer une part plus ou moins importante de l'activité globale de programmation. Les méthodes qui constituent une aide pour cette partie du problème sont fortement liées au domaine propre du problème (on en trouve le plus d'exemples dans le domaine de la gestion).

Dans les méthodes tournées vers la formation et l'enseignement de la programmation un développement important est accordé au processus qui suit la spécification. La plupart s'appuient sur les trois notions fondamentales de : programmation structurée, programmation modulaire et programmation descendante (Rogalski et al.).

Leur fonctionnalité réelle dépend de la classe du problème de programmation : selon que "programmer c'est faire faire un calcul" ou "programmer c'est concevoir un algorithme" (C.Pair), l'aide méthodique ne portera pas sur les mêmes difficultés.

1. La situation limite pour le premier type de problèmes est celle où :

a) les objets du problème sont déjà formalisés ou formalisables immédiatement dans une représentation connue des élèves (mathématique en général) qui joue le rôle de précurseurs,

b) un algorithme est déjà connu pour un "traitement à la main". Le problème va être d'articuler les primitives : enchaînement, traitements conditionnels et itération (programmation structurée).

Il s'y ajoute la nécessité de passer des représentations initiales aux objets informatiques pertinents : cela suppose "des connaissances minimales sur les concepts informatiques et sur les contraintes liées à l'exécution sur un dispositif informatique donné." (Rogalski, Vergnaud, 1987).

L'enseignement de méthodes pour traiter cette phase du problème de programmation apparaît ici étroitement lié à celui même des concepts informatiques de base. Ainsi J. Arsac (1980) explicite les phases de la construction d'une itération dans ses "Premières leçons de programmation" en s'appuyant sur la notion de situation (ou d'état de la situation). Nous reviendrons plus loin sur l'analyse des obstacles rencontrés dans l'appropriation par les élèves de la méthode proposée.

L'introduction des schémas proposés par C. Pair suppose des acquis dans le domaine de la structuration des données, permettant de donner du sens au codage proposé. Nous manquons de connaissances systématiques sur les difficultés cognitives dans ces problèmes de structuration des données pour analyser les problèmes cognitifs posés par la mise en oeuvre d'une méthode. Un précurseur sur lequel peut s'appuyer l'enseignement est le "traitement à la main": il donne du sens aux différentes primitives (structures séquentielle, conditionnelle, itérative) et joue aussi un rôle producteur. Mais ce précurseur joue aussi un rôle réducteur car les stratégies spontanées des débutants consistent à générer alors le programme dans une situation d'exécution mentale, qui concerne nécessairement un calcul, et non un ensemble de calculs. De plus cette exécution mentale est liée au système de traitement de l'humain qui peut utiliser présupposés, anticipation et parallélisme.

Par ailleurs, un renforcement du modèle "spontané" d'exécution mentale produira presque nécessairement plus tard des obstacles pour l'acquisition et l'utilisation des méthodes de programmation descendante, car la situation d'exécution mentale se fait au niveau du "détail" et non des grandes fonctions à remplir par le programme.

La méthode développée par J. Arsac rompt avec l'usage de telles "exécutions mentales" ; mais globalement l'enseignement initial de l'informatique n'introduit-il pas un environnement d'exécution ?

Dans ce contexte, l'utilisation des situations exige une rupture, doublement difficile pour les élèves, car elle concerne à la fois des éléments du contrat didactique et des notions conceptuelles.

2. Dans le second type de problèmes il faut "dans un premier temps, exprimer l'algorithme sur les objets du problème (...) ; cela introduit des fonctions et des procédures, et on choisit alors une représentation des objets qui favorise l'expression de ces fonctions et procédures" (C. Pair). Les méthodes qui visent à aider à une programmation descendante et modulaire peuvent être adaptées à ce type de problème. Elles peuvent être prospectives : partant des données initiales vers les résultats à obtenir, ou rétrospectives : remontant des résultats à obtenir vers les données.

La description succincte de l'activité de programmation dans ce type de problèmes montre que des connaissances informatiques sont supposées acquises (fonctions, procédures, relations avec les structures de données possibles) : les méthodes liées au type 2 concernent donc essentiellement la gestion de connaissances, et non leur acquisition.

Ces méthodes possèdent en général les fonctionnalités suivantes :

- modélisation du problème initial
- hiérarchisation et organisation temporelle des phases de résolution
- communication entre programmeurs

(ces deux dernières fonctionnalités répondent à des problèmes d'organisation du travail)

- ouverture des possibles, en différenciant les fonctions à remplir par la solution des réalisations elles-mêmes (en programme)
- évaluation objective des solutions possibles et optimisation du choix
- réalisation (et modification possible) de la solution retenue.

(Rogalski et al.)

La transposition de ces méthodes (issues des contraintes de pratiques professionnelles) dans des situations d'enseignement est une opération délicate. Pour que la méthode apparaisse comme un outil, qui aide à traiter des problèmes, il faut que le problème soit assez complexe ; pour que cet outil fonctionne avec des élèves dont les acquisitions sont encore en cours de maturation, il faut que la complexité du problème soit "gérable" dans les conditions de travail de la classe (dans laquelle l'organisation du travail commun n'est pas une motivation familière).

En fait, un pas conceptuel à franchir pour aborder des problèmes de type 2, autant que pour acquérir des méthodes qui leur sont attachées, est de passer d'un modèle

d'exécution à une conception du programme comme une fonction qui fait passer des données au résultat :

$$\mathcal{P} : \mathcal{D} \longrightarrow \mathcal{R}$$

et à se poser le problème de réalisation de cette fonction globale, soit en définissant des états intermédiaires, soit en raffinant la définition des données et du résultat.

Est-il possible de franchir ce pas sans avoir déjà "manipulé" comme outils des fonctions et des procédures qui produisent des passages bien identifiés quant à leur contenu, entre des données et des résultats ? Pour que de tels outils ne soient pas artificiels, il est nécessaire que leur élaboration elle-même soit assez complexe pour que l'économie cognitive et de temps compense l'effort que doit faire l'élève pour intégrer des éléments de type "boîtes-noires" dans le programme qu'il construit (ces éléments interdisant par ailleurs le recours à l'exécution mentale).

QUELLES SITUATIONS DIDACTIQUES POUR FAIRE ACQUÉRIR DES MÉTHODES EN PROGRAMMATION ?

Nous avons dégagé deux types de problèmes et de méthodes associées. Une première conséquence d'une telle classification est que différents types de situations didactiques, associées probablement à des moments différents de l'enseignement de l'informatique, vont devoir intervenir pour faire acquérir par les élèves des méthodes de programmation qui les aident effectivement dans la réalisation de programmes "satisfaisants".

Un premier obstacle apparaît dans l'acquisition de l'itération, et de méthodes appropriées à l'écriture de programmes itératifs ou récursifs. On peut exprimer cet obstacle sous deux formes différentes. D'une part la maîtrise de la dialectique entre exécution d'actions (ou "calculs") avec son caractère dynamique, et succession de situations (ou "relations") avec un caractère statique de la situation semble au coeur de l'acquisition des constructions itératives ou récursives.

D'autre part, la variable informatique doit être conçue comme une fonction de l'exécution ; une telle dualité variable/fonction n'est pas spécifique à l'informatique mais elle n'est jamais apparue dans le "passé cognitif" des élèves (qui maîtrisent en général fort mal à l'issue de l'enseignement secondaire les relations variable/fonction). La notion d'invariant qui est au coeur de la présentation de l'itération par J. Arsac peut difficilement recevoir une signification sans la maîtrise de cette double perspective.

Nous avons utilisé le scénario suivant avec des étudiants de psychologie (juste "alphabétisés" en programmation ou "naïfs").

L'écriture de l'état de l'ensemble des variables d'un programme centré sur une structure itérative, à la fin de chaque instruction et à la fin de l'exécution du corps de la boucle peut fournir des bases :

1) pour distinguer des variables qui gardent leur valeur tout au long de l'exécution de celles dont la valeur est modifiée au cours de l'exécution de la boucle,
2) pour initier un travail sur : a) l'identification empirique de relations invariantes entre valeurs de variables et b) la validation de ce constat par un raisonnement portant sur les transformations des variables dans le corps de la boucle.

La comparaison de ce travail effectué sur deux programmes répondant à un même problème (ici : calculer un produit d'entiers positifs par additions réitérées) permet d'introduire l'existence d'invariants différents liés à des corps de boucle différents.

La justification d'un invariant "paradoxal" oblige à travailler sous des hypothèses du type "supposons que nous soyons à la ième exécution du corps de boucle". Un tel travail "sous hypothèse" constitue probablement une autre des difficultés majeures des méthodes d'élaboration de programmes itératifs (ou récursifs) : il peut être productif pour les élèves de le rencontrer sous des formes différentes de difficultés variables, liées à une même problématique.

Peut-être des activités de cet ordre, conduites avec des élèves pour lesquels des textes de programme seraient des objets déjà un peu familiers (même quand ils ne les ont pas produits eux-mêmes) faciliteraient-elles la phase d'écriture de programmes itératifs à l'aide de l'outil -puissant- d'analyse de situations ?

Un second obstacle, davantage lié aux problèmes et méthodes de type 2, est celui du changement de perspective : l'élève ne doit plus (seulement) concevoir le programme comme décrivant des calculs mais comme exprimant une fonction. Un tel changement de perspective doit s'accompagner du changement de point de vue : de la question "comment agit le programme" l'élève doit en passer à la question "que réalise le programme" (à quel objectif répond- il ?). Nous avons déjà indiqué plus haut que le contexte "d'exécution mentale" contribuait à renforcer cet obstacle car il est centré sur le "comment". Nous avons également fait l'hypothèse que l'utilisation de fonctions (et procédures) comme "briques" pour élaborer des programmes conduisant les élèves à se centrer sur la question du but du programme, situations didactiques impliquant un partage de tâches de programmation, l'identification de programmes préexistants utiles, et leur intégration dans le programme à réaliser permettraient de donner un sens et une motivation à de telles activités.

Le travail sur un projet de (petit) groupe à l'issue de ces activités (ou en parallèle avec elles) pourrait enfin être conçu non seulement comme motivant pour les élèves, mais aussi comme une transposition productive des situations de pratique informatique qui ont conduit à construire des méthodes de programmation, et à décider de les enseigner...

Conclusion

Les difficultés d'enseignement et d'acquisition de méthodes en programmation apparaissent liées à des obstacles conceptuels profonds. L'ensemble du contexte d'enseignement interagit avec les effets de situations destinées à "supporter" un enseignement de telles méthodes. L'élaboration de situations didactiques doit donc être considérée dans la durée, longue, de l'initiation à l'informatique. Il faut sans doute rejeter l'illusion qu'il serait possible de contourner ces obstacles par des artifices didactiques. Nous pensons qu'au contraire il faut les identifier et les faire affronter aux élèves ; peut-être devrait-on aussi les conduire à identifier eux-mêmes l'effet de ces obstacles dans leurs échecs ou leurs difficultés ?

Une limite doit être relevée dans les esquisses d'analyse et de solutions possibles : elles concernent un contexte de programmation "impérative". Beaucoup de recherches sont à développer sur les difficultés propres et les effets producteurs possibles d'autres contextes d'enseignement, en jouant sur les propriétés "procédurales"/"fonctionnelles" de différents langages de programmation.

BIBLIOGRAPHIE

- J. ARSAC (198O). *Premières leçons de programmation*, Cedic- Nathan, Paris.

- J. ARSAC (1987). *Des pédagogies pour l'option informatique*, Pratiques et savoir-faire des élèves de l'option informatique, 4, 1-13.

- O.J. DAHL, E.W. DIJKSTRA, C.A.R. MOARE (1972). *Structured programming*, Academic Press, London.

- A. DUCRIN (1984). *Programmation*, Dunod, Paris.

- A. GRAM (1986). *Raisonner pour programmer*, Dunod, Paris.

- J.M. HOC (1978). *Etude de la formation à une méthode de programmation informatique*, Le Travail Humain, 4O,15-28.

- C. PAIR (1988). *Je ne sais (toujours) pas enseigner la programmation*.

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

- A. ROBERT, J. ROGALSKI R. SAMURCAY (1987). *Enseigner des méthodes,* Cahiers de Didactique des Mathématiques, 38, IREM, Université Paris VII.

- J. ROGALSKI (1987a). *Acquisition de savoirs et savoir-faire en informatique*, Cahiers de Didactique des Mathématiques, 43, IREM, Université Paris VII,

- J. ROGALSKI (1987b). *Acquisition de savoirs et savoir-faire,* pratiques et savoir-faire des élèves de l'option informatique, 14-22.

- J. ROGALSKI, R. SAMURCAY, J.M. HOC (à paraître). *L'apprentisage des méthodes de programmation comme méthode de résolution de problème*, Le Travail Humain".

- J. ROGALSKI, G. VERGNAUD (1987).*Didactique de l'informatique et acquisitions cognitives en programmation*,Psychologie Française, 4, 267-273.

- J. ROGALSKI (1986). *Pour une pédagogie de l'informatique*, Bulletin de l'EPI n° 42, juin 1986, 105-109