



# L'informatique : une nouvelle façon de penser

La réponse  
de J. ARSAC

*Un professeur doit être patient. S'il constate qu'il n'a pas été compris, il recommence, essayant d'être plus clair et plus précis. C'est ce que je me vois contraint de faire une fois de plus, après la publication de l'article de M. J. L. Lemoigne (Terminal 19/84, n° 11, octobre 82). Je ne peux laisser sans réponse les accusations qu'il porte contre moi. Il me faut aussi essayer de faire percevoir au lecteur, en peu de mots, en quoi consiste cette nouvelle façon de penser en informatique.*

## Une conception impérialiste de l'informatique ?

M. Lemoigne me reproche d'être impérialiste. Je reconnais volontiers que lorsqu'on défend une cause qui vous tient à cœur, on parle avec chaleur et passion, oubliant d'émailler le discours de place en place de précautions oratoires rappelant que d'autres pensent autrement. Mais je peux opposer les faits à l'accusation gratuite de M. Lemoigne. Dans ce domaine de l'enseignement de l'informatique où je serais impérialiste, j'ai eu à démarrer l'enseignement de l'option informatique en classe de seconde, dans la ligne du rapport de J.C. Simon (jamais Simon, ni moi-même n'avons préconisé l'enseignement obligatoire de la **théorie** de la programmation, contrairement à ce qu'affirme M. Lemoigne). Il m'eût été facile d'imposer ma façon d'enseigner. Au lieu de cela, cinq centres de formation des maîtres ont été créés, d'où sont sorties cinq pédagogies différentes...

M. Lemoigne récusé ma participation à l'enseignement parce que je « *n'ai pas su comprendre en 1960, la supériorité conceptuelle profonde sur l'américaine IBM 1130 de la française CAB 500* ». Nous avons pourtant enseigné et utilisé le PAF à cette époque, et la CAB 500 m'a

### UN DEBAT PUBLIC CAPITAL

Vous avez publié dans le numéro 11 de la revue *Terminal* 19/84 un article de M. Lemoigne qui ne m'épargne guère. Les critiques personnelles ne me tracassent pas trop, encore qu'elles soient contraires aux faits, et que M. Lemoigne ait pratiqué l'amalgame pour faire croire que j'étais responsable d'opérations avec lesquelles je n'ai rien à voir.

**Mais il y a actuellement un débat public capital sur la vraie nature de l'informatique.** Est-elle, comme le pensent un certain nombre de gens, en général hors de l'enseignement, une technique triviale qui s'apprend en trois semaines (B. Lussato) ou une science dont l'apprentissage demande plus de travail, et une pédagogie soignée ? C'est un grave débat car il modifie les perspectives en matière d'informatisation de la société. Il est ridicule d'en faire un conflit entre l'ENS et l'X. C'est un choix d'idées, pas d'hommes, même s'il est difficile d'éviter les arguments *ad hominem*. Et s'il est vrai que j'aurais dû m'abstenir de signer la lettre du *Nouvel Observateur*, je ne pense pas que M. Lemoigne puisse me donner de leçons, car il me traite encore plus mal, sous des dehors hypocrites (J. Arsac, le plus éminent, etc.). J'ai essayé dans ma réponse d'éviter de tels arguments, et de présenter les idées auxquelles je crois, et pourquoi j'y crois.

J. ARSAC

servi de modèle pour le système Forceps que j'ai écrit en 1968 (10 000 instructions, pas deux millions...) et dont l'architecture se retrouve intégralement dans le système LSE écrit plus tard par l'équipe du professeur Hebenstreit. Quant à m'accuser de « *proposer (1973) comme seul objectif digne de la recherche l'écriture d'un système d'exploitation de deux millions d'instructions* », c'est faire preuve d'une ignorance totale de mes travaux et publications. Depuis 1971, je me suis consacré exclusivement à la méthodologie de la programmation : langages sans instruction d'affectation, transformation de programmes, méthodes constructives en programmation, synthèse de programmes, vulgarisation de ces idées nouvelles...

## Une définition marginale de l'informatique

Selon M. Lemoigne, « *l'informatisation, science des calculateurs arithmétiques et logiques automatisables, ne cautionne que très marginalement mon discours* ». Je reviendrai plus loin sur cette affaire de caution. Je voudrais signaler ici que M. Lemoigne utilise une définition très marginale de l'informatique. Déjà en 1968, Perlis notait que l'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes ou la thermodynamique n'est celle des thermomètres. L'académie française la définit comme « *la science du traitement rationnel, notamment par machines automatiques, de l'information considérée comme le support des connaissances humaines...* ». C'est le point de vue que j'ai défendu dans mon ouvrage *La science informatique* (Dunod, 1970).



## Ancienne et nouvelle façon de pensée

Mais laissons-là ces accusations, au fond, très secondaires, pour entrer au cœur du débat. Y a-t-il une nouvelle façon de penser en informatique ? Voici d'abord ce que j'appelle l'ancienne façon de penser. Ce que l'on m'a enseigné en 1956, ce que j'ai ensuite enseigné jusqu'en 1972, ce qui s'enseigne encore, notamment dans des ouvrages de parution récente, c'est une brève présentation de la structure des ordinateurs, suivie d'une longue description d'un langage de programmation : le nom d'une variable commence par une lettre et est formé d'au plus cinq caractères, lettres ou chiffres... Les constantes numériques s'écrivent... L'instruction SI... L'instruction FAIRE, etc. Pour résoudre un problème, on dessine un ordigramme, on rédige un programme, puis on le met au point sur des données tests. Mais on ne dit rien sur la façon d'inventer l'ordigramme, ni comment expliquer pourquoi le programme produit les résultats attendus. Ceci reste affaire de savoir intuitif du programmeur. (J'aurais aimé savoir si c'est là cette méthode analytique, réductionniste et mutilante dont parle M. Lemoigne). La caution m'est refusée parce que cette méthode a produit de bons résultats. Il est très vrai qu'il y a d'énormes programmes qui fonctionnent bien : systèmes d'exploitation, compilateurs, systèmes ban-

caires ou d'assurance, réservation de places, programmes de paie... Mais tout de même, depuis le colloque tenu à Monterey en 1973 sur le coût élevé du logiciel, on n'a cessé de dénoncer le prix exagéré du logiciel, et ses délais de fabrication : pendant que le prix du matériel diminuait de façon fantastique, celui du logiciel croissait comme celui des salaires, parce que le programmeur de 1982 n'en sait pas plus que celui de 1960. Le savoir faire intuitif est incommunicable.

### La crise du logiciel

A côté de brillantes réussites, combien d'échecs ? Je parle d'expérience de ces petites entreprises qui ont acheté un micro-ordinateur et son logiciel pour s'apercevoir qu'il contient des erreurs et ne rend pas les services promis. J'ai dû intervenir dans un litige entre une société de service et son client pour un programme faux. Le programmeur connaissait fort bien Basic, beaucoup mieux que moi, mais il ne savait pas faire un tri, ni une fusion de fichiers, ni se servir d'index sur un fichier... Quant à son style de programmation, je n'ose en parler (programmes illisibles). Si l'on récuse mon témoignage, on ne pourra refuser celui du département de la Défense des Etats-Unis qui déclarait, il y a cinq ans, dans les attendus de l'appel d'offre pour un nouveau langage (ADA) que s'il réduisait seulement de 1 % les erreurs de programmation commises pour son compte, il économiserait 25 millions de dollars par an. Cela me suffit comme caution.

### L'instruction d'affectation

La difficulté en programmation vient de l'instruction d'affectation, dont le modèle est :  $X = X + Y$ . Élémentaire, un enfant de sept ans comprend cela : on ajoute la valeur de Y à celle de X pour obtenir la nouvelle valeur de X. Alors pourquoi la suite :  $X = X + Y$  ;  $Y = X - Y$  ;  $X = X - Y$  est-elle si difficilement compréhensible ? On a mis fort longtemps à réaliser que là était la source de toutes les difficultés, le plus grand facteur d'obscurité des programmes.

De fait l'instruction d'affectation change la valeur d'une variable, elle est de nature transformationnelle. Dès lors, un programme dit la suite de transformations qui feront passer de la situation initiale (celle des données) à la situation finale (celle des résultats). Or, cette suite de transformations n'est pas du tout parlante. Ce que l'on suit aisément, c'est la séquence des situations qu'elle engendre. Une façon de dire comment un programme marche, c'est de présenter la suite de situations qu'il engendre. Mais l'expérience montre que c'est très difficile à faire dès qu'il y a itération.

### Donner la priorité à la situation sur l'action

C'est pourquoi, on en est venu à dire (*la construction de programmes structurés*,

J. Arsac, Dunod 1977, puis *Premières leçons de programmation*, J. Arsac, Cedic-Nathan, 1980) qu'il faut que le programmeur auteur du programme donne lui-même cette suite à la création du programme. Mieux, il faut construire le programme en partant de la suite de situations à réaliser. Ainsi se trouve renversée la perspective habituelle. On m'a appris à programmer en cherchant quelle doit être la prochaine action à faire. La nouvelle façon de penser, c'est de donner priorité à la situation sur l'action. La question n'est pas « *que vais-je faire ?* », c'est « *où en suis-je ? Quelle est la situation que je veux atteindre ?* ». L'action s'en déduit en général simplement.

### Une façon puissante de construire les programmes itératifs

Bien sûr, l'itération complique un peu les choses. Parce que l'exécution du programme repasse plusieurs fois par le même point, on retrouve plusieurs fois la même situation. D'où une façon très générale et très puissante de construire des programmes itératifs :

— imaginer une situation générale. Le plus souvent, on supposera que l'on a fait une partie du travail. Je dois trier un vecteur de  $n$  éléments ; supposons que j'ai trié les  $i$  premiers éléments ;

— voir si c'est fini. On regarde à quelle condition le but est atteint (dans le cas évoqué ci-dessus, c'est fini si  $i = n$ ) ;

— si ce n'est pas fini, il faut se rapprocher de la solution. ce faisant, on s'éloigne de la situation générale. Il faut la rétablir. Dans le cas du tri, on peut se rapprocher de la solution en étendant de 1 unité la partie triée. On a alors une partie triée de longueur  $i + 1$ . On met  $i + 1$  à la place de  $i$ .  $i = i + 1$  ;

— reste à démarrer en trouvant une situation initiale, facile à atteindre, cas particulier de la situation générale. Dans le cas du tri, une partie réduite à un seul élément est triée,  $i = 1$ .

On fait apparaître ainsi un nouveau problème : mettre un élément à sa place dans un tableau trié. C'est un problème plus simple. On le traite à son tour de la même façon. C'est ce que l'on appelle « *la programmation descendante* ».

### Descartes l'a dit et il y a bien longtemps

Il est absolument évident (me serais-je à ce point mal exprimé ?) que cette façon nouvelle de travailler, cette façon nouvelle de penser en informatique n'est pas la découverte d'une façon nouvelle de penser. L'homme n'a pas attendu 1975 pour dire qu'il faut réfléchir avant d'agir. L'informatique n'a pas le privilège de dire que l'action à prendre se déduit de la connaissance de la situation présente et de la situation visée. L'extraordinaire c'est plutôt que cette façon de penser ait été si longue à mettre en place en infor-





matique, tant elle va de soi. De même, la programmation descendante est un acquis important de la science informatique pour la maîtrise de la complexité. Mais elle n'est pas nouvelle : Descartes avait dit cela il y a bien longtemps...

On va sans doute me dire que j'enfonce des portes ouvertes, que tout le monde sait et pratique cela. Mon expérience d'enseignant dans les milieux les plus divers, et notamment chez les professionnels de l'informatique, m'a convaincu du contraire. Je retrouve la même chose dans les livres couramment publiés : dans l'un d'eux, sorti il y a moins de trois ans, l'auteur présente un programme de

tri, explique sur un exemple comment il marche : le plus grand élément monte, par transposition successives vers le haut du vecteur ; or l'ordinogramme et le programme associé dont au contraire descendre le plus petit vers le bas. Je me suis livré à des expériences avec les étudiants du DEUG scientifique, étudiant dans leur deuxième année après le bac de terminale C, réputé le plus sélectif. Pour la plupart, non seulement la façon de penser qui donne le pas à la situation sur l'action ne leur est pas familière, mais encore ils ont du mal à distinguer les deux notions. A la question : « *En tel point du programme, quelle est la situation atteinte ?* », nombreux répondent : « *On va faire une boucle de 1 à n* ». Ils définissent la situation par l'action à venir.

### L'enseignement pratique la vieille façon de penser

Je le répète, pour qu'on ne me fasse pas dire le contraire de ce que je pense : il y a une nouvelle façon de construire un programme, en jalonnant le parcours entre la situation initiale et la situation finale de situations intermédiaires, les actions à prendre pour passer de l'une à l'autre s'en déduisant. Cette façon de penser n'est pas une conquête de la

science informatique, elle est probablement aussi vieille que l'humanité. Néanmoins, il se trouve que l'enseignement secondaire français n'en favorise pas toujours la pratique chez les élèves.

C'est pourquoi, j'ai défendu avec J.C. Simon, l'idée que l'enseignement de l'informatique aux lycéens pouvait contribuer à l'enrichissement de leur façon de penser. Encore une fois, non pas parce que nous avons inventé une nouvelle façon de penser, mais parce qu'il y a ici pratique d'une vieille façon de penser, pas toujours aussi présente qu'on ne pourrait le souhaiter. On nous a objecté que travailler ainsi, de façon méthodique, allait à l'encontre de la créativité chez les jeunes, que ce discours serait soit difficile à tenir, soit ennuyeux. C'était raisonner en programmeur déformé par vingt ans de pratique (le public le plus difficile à qui il m'ait été donné d'enseigner, est celui des programmeurs d'un grand constructeur : comment cet universitaire dans sa tour d'ivoire pourrait-il nous apprendre quelque chose, à nous, praticiens réputés ?).

### On juge un arbre à ses fruits

C'était aussi compter sans les talents pédagogiques des professeurs de lycée. Ils ont adapté la méthode aux jeunes non scientifiques (O. Arsac-Mondou, C. Bourgeois-Camescasse, M. Gourtay : *Premier livre de programmation*, Cedric-Nathan, 1982). Ils ont enseigné cette méthode aux jeunes. Le résultat : alors que dans les clubs informatiques, nombreux sont ceux qui abandonnent rapidement laissant l'ordinateur aux petits malins, la majorité des élèves de seconde ont ainsi pu concevoir des programmes, avec des temps de mise au point très courts (sans rapport avec ce qui se pratique tous les jours dans la profession).

Je n'ai aucune prétention impérialiste. Je n'ai jamais empêché qui que ce soit d'enseigner comme il le voulait. Il se trouve que la programmation a évolué, passant du savoir-faire de l'artisan à la méthode de l'ingénieur (en 1968, D. Knuth publiait *The art of computer programming*. En 1981, David Gires publie *The science of programming*). Cette mutation me paraît importante. Je crois de mon devoir de la mettre à la portée du plus large public possible parce que l'informatique touche de plus en plus de gens. Je n'ai pas d'idolâtrie pour la nouveauté, et je me méfie systématiquement des méthodes nouvelles : « *Oui, cela était ainsi autrefois, mais nous avons changé tout cela, et nous faisons maintenant la médecine d'une méthode toute nouvelle* ». On juge un arbre à ses fruits. J'ai étudié le fruit de la programmation « empirique ». J'ai observé ceux des méthodes nouvelles, et j'ai choisi. ■

Jacques ARSAC

Directeur de la section Informatique de l'ENS, octobre 1982.

