LA DIDACTIQUE DE L'INFORMATIQUE : UN PROBLÈME OUVERT ?

Jacques Arsac

Professeur à l'Université P. et M. Curie
Chargé d'une mission d'inspection générale

LA DIDACTIQUE DE L'INFORMATIQUE : UN PROBLÈME OUVERT ?

Jacques Arsac

Professeur à l'Université P. et M. Curie
Chargé d'une mission d'inspection générale

1. L'ÉMERGENCE DE LA SCIENCE INFORMATIQUE.

Dès que l'on eut pris conscience qu'un ordinateur pouvait faire autre chose que des additions ou des divisions, dans le milieu des années 50, bien avant que le mot "informatique" ne fut inventé (Philippe Dreyfus; 1962), l'idée s'imposa que l'ordinateur avait quelque chose à faire à l'école. Cela commença par les "machines à enseigner". Au colloque IFIP de Munich, en 1962, un conférencier annonçait son projet gigantesque : on allait faire des machines qui enseigneraient mieux que les professeurs (c'est tout juste si le conférencier n'ajoutait pas : et ce n'est pas difficile !). Mais il tenait à rassurer les professeurs : ils ne seraient pas mis au chômage pour autant, on aurait toujours besoin d'une présence affective auprès des élèves ... J'ai tant entendu de ces annonces fracassantes et jamais suivies d'effet que je suis devenu très prudent : je ne crois que ce que je vois fonctionner. Sous l'effet des premiers échecs, le projet s'est assagi, on a abandonné l'idée de machines autonomes pour celui de machines assistant le professeur ou l'élève (EAO), encore que certains des discours actuels sur l'enseignement intelligemment assisté par ordinateur ne ressemblent étrangement à ceux des années 60. Mais là n'est point le thème de ce colloque.

Dès le début des années 60, les professeurs qui enseignaient les ordinateurs, leur structure et leur architecture, leur programmation, leurs applications dans les domaines numériques et non numériques prirent conscience qu'une science nouvelle était en gestation (*computer science* aux USA, *datalogie* en Scandinavie, *informatique* en France) et que cela posait de sérieux problèmes d'enseignement. Quels étaient les contours de cette science ? Que fallait-il enseigner ? A qui ? Avec quels prérequis ? L'IFIP se dota en 1964 d'un comité technique pour l'enseignement (IFIP TC3) dont j'eus l'honneur de faire partie jusqu'à ce que les événements de 1968 me contraignent à restreindre mes activités internationales. Ce comité était convaincu que l'informatique n'était pas qu'une affaire de

spécialistes, mais qu'elle devait être mise à la disposition d'un large public. Elle avait sa place dans l'enseignement général.

Ainsi fut organisé en 1970, sous l'égide de l'OCDE, le colloque de Sèvres [SEV70]. Ses travaux furent prophétiques : il recommandait en effet que l'informatique soit introduite dans l'enseignement général non parce que l'informatique est un phénomène technique d'une grande importance économique, ou parce qu'elle a d'importantes retombées sur l'organisation même de la société et sur nos manières de vivre, ni pour satisfaire aux besoins de la profession, mais parce que "la pratique de la programmation développe des aptitudes algorithmiques, organisationnelles et opératoires".

Je les dis prophétiques parce que ce que l'on savait de la programmation à l'époque ne permettait pas d'atteindre de tels objectifs. La programmation était un empirisme, ou, dans le meilleur des cas, un art ("The art of computer programming", D.E. Knuth, 1968 [KNU68]). Comment enseigner cela, et quel rapport avec une science, ou plus simplement avec des objectifs de formation de l'esprit ? Le colloque tenu à Monterey en 1973 sur "le coût élevé du logiciel" [MON73] mit en évidence les graves défauts de l'empirisme : à cause de lui, le logiciel coûte beaucoup trop cher, il n'est pas fiable, pas transportable, pas modifiable, rarement produit dans les délais prévus ... La seule réponse possible était dans le développement d'une théorie. Les chercheurs s'étaient déjà mis au travail. Peter Naur [NAU66], puis Robert Floyd [FLO67] avaient posé les bases des preuves de programmes. Dijkstra avait condamné les instructions de branchement [DIJ68]. Naur avait introduit la programmation descendante [NAU69], et Dijkstra la programmation structurée [DDH72].

A la suite de ces travaux de recherche, la programmation changea de nature. Elle approcha du statut de discipline scientifique (*A discipline of programming*, Dijkstra [DIJ76]), puis fut reconnue comme science (*The science of programming*, Gries [GRI81]). Ces méthodes scientifiques changèrent la façon de programmer, et l'on vit apparaître la "méthodologie de la programmation" comme champ de recherche, avec des méthodes de programmation comme résultat concret. Les premières naquirent dans le cadre relativement restreint de l'informatique de gestion [WAR75], [JAC75]. Des méthodes de portée plus générale, et partant moins précises dans leurs détails, furent proposées pour aider à la création de nouveaux algorithmes ([DIJ76], [ARS77], [ARS80], [GRI81], [DUC84]). Tout ceci ne pouvait manquer d'avoir un impact sur l'enseignement.

2. QUEL ENSEIGNEMENT DE L'INFORMATIQUE

D'autant que dans le même temps le succès de l'informatisation de nombreuses activités impose l'enseignement de quelque chose de l'informatique à des publics de plus en plus variés. Les mathématiciens redécouvrent l'importance de l'algorithmique et mettent de plus en plus d'informatique dans leur programmes. Les chercheurs scientifiques et ingénieurs de toutes disciplines ont besoin d'informatique, mais sont-ils des utilisateurs

de bibliothèques faites par d'autres, ou ont-ils à être créateurs ? Doivent-ils faire une confiance totale à ces bibliothèques, comment peuvent-ils en contrôler les résultats ? Le même problème se pose pour les économistes, les géographes, et commence à apparaître pour les littéraires. Qui pourrait dire aujourd'hui que l'informatique ne le concerne pas ?

En période de chômage, on a vu des secrétaires abandonner leur poste plutôt que de se trouver affrontées à un traitement de texte. L'ordinateur fait peur à certains. Une sensibilisation est nécessaire pour éviter ce genre de réactions. Pour qu'elle touche tout le monde, c'est à l'école qu'il faut la faire : ainsi est né le plan "informatique pour tous". Mais jusqu'où doit-il aller ? Savoir mettre l'ordinateur en route, glisser la disquette dans le bon sens dans le lecteur, appuyer sur quelques touches, ou au contraire bien connaître le dernier traitement de textes et le dernier tableur ? N'est-ce pas un investissement perdu : avant que l'élève ne sorte du lycée, ces produits auront disparu du marché... L'élève doit-il savoir quelque chose de l'ordinateur et de la façon dont on lui fait réaliser une tâche, ou peut-on considérer matériel et logiciel comme une boîte noire ? Quelle compétence est demandée au professeur ? Doit-il être un expert en traitement de texte, tableur ou gérant de bases de données ? Doit-il savoir de l'informatique ? Doit-il être compétent en programmation ?

Le débat est ouvert : quelle informatique pour de futurs utilisateurs ? Quelle sensibilisation pour tous ? On a utilisé des comparaisons : il n'y a pas besoin de savoir comment est fait un moteur de voiture pour bien conduire. Il n'y a pas besoin d'avoir étudié les équations de Maxwell pour regarder la télévision ... Mais comparaison n'est pas raison, dit la sagesse populaire. On le constate déjà : l'informatisation s'accompagne d'effets secondaires néfastes. Qui ne s'est vu répondre : "nous n'y pouvons rien, c'est la faute de l'ordinateur". Sans un minimum de connaissance de cette machine, de la façon dont elle travaille et dont on la commande, qui peut rejeter cette excuse qui cache soit l'incompétence de celui qui se sert d'un logiciel, soit son peu d'empressement à se lancer dans des manoeuvres plus complexes, soit l'incompétence de ceux qui ont programmé le système ?...

Les médias nous annoncent régulièrement l'apparition de nouveaux produits fantastiques. Il y a peu, un journaliste racontait comment la France était en train de perdre la bataille du téléphone, une firme anglaise venant de mettre au point un téléphone polyglotte : vous parlez en français, et votre correspondant vous entend en anglais. Suivait une interview d'un ingénieur, expliquant que le système possédait un stock de 400 phrases françaises avec leur équivalent anglais. Un système déjà fort complexe de reconnaissance des formes tentait de reconnaître dans ce que dit le locuteur une de ces phrases, et la remplaçait alors par son équivalent anglais. L'ingénieur ajoutait qu'avec cela, il pensait que l'on avait 80% de chances de pouvoir retenir une chambre d'hôtel ! Un autre journaliste, vantant les systèmes experts, expliquait que l'un d'eux pouvait reconnaître deux mille espèces de champignons en quatre questions ! On peut accuser ces journalistes de manquer d'esprit critique. Si on venait leur expliquer que l'on peut trouver du pétrole au

fond de l'océan à partir d'un avion en vol, ils ne manqueraient de s'esclaffer, voire de crier au scandale : des avions renifleurs, pensez-donc ? Alors pourquoi ne peuvent-ils faire de même, s'agissant d'informatique ? Mais peut-on vraiment les accuser ? Ont-ils eu la possibilité d'apprendre quelque chose de l'informatique dans leurs études ? De se recycler plus tard ? Faut-il que le citoyen soit le jouet des médias ? Comment peut-il se défendre ? "La seule réponse à la superstition, c'est l'éducation" a dit un jour Jules Ferry. Quelle éducation pour former l'esprit critique dans un monde en cours d'informatisation ?

Tout ceci pose de manière grave le problème du contenu des enseignements d'informatique pour tout public autre que les futurs professionnels de cette discipline (en admettant que pour eux la question soit résolue, ce qui paraît tout de même une hypothèse raisonnable). Faut-il apprendre aux gens le maniement d'outils dont on connait la rapide obsolescence ? Peut-on estimer que des fonctions essentielles ont été mises en évidence qui perdureront dans les futurs produits, et qu'à travers les réalisations actuelles, ce sont elles que l'on enseigne ? Est-ce bien cela qui est actuellement pratiqué ? Faut-il au contraire renoncer à l'idée de boîte noire, jusqu'où faut-il aller ? Quelle quantité de programmation est nécessaire pour comprendre ce que peuvent, et peut-être ne peuvent pas les ordinateurs ?

Ceci débouche à nouveau sur l'enseignement de la programmation. C'est un débat que l'on ne peut esquiver, ne fut-ce qu'à cause du rapport que déposa en 1979 Jean-Claude Simon, à la demande du Président de la République [SIM79]. Il reprit en effet les recommandations du colloque de Sèvres, demandant que l'informatique soit enseignée au lycée pour des raisons de formation de l'esprit. A la suite de quoi, j'ai été appelé à créer l'option informatique des lycées, dont j'ai dû définir les horaires, les programmes, la façon de former et recruter les professeurs. Après sept années de fonctionnement, l'option continue sur le schéma initial, ayant perdu en 1985 son caractère expérimental, et ayant donné lieu pour la première fois en 1988 à une épreuve de baccalauréat. Or les problèmes de pédagogie sont d'autant plus graves que l'on s'adresse à des élèves plus jeunes : "pédagogie" vient du grec "pais" qui veut dire "enfant". Peut-on encore parler de pédagogie quand on s'adresse à des adultes en formation permanente ? Sans jouer sur une étymologie qui n'a peut-être plus toute sa force, il est certain que les plus jeunes sont les moins aptes à faire face aux déficiences de l'enseignement : les étudiants ont les reins plus solides... Or, force est de le constater, les travaux de recherche que je mentionnais plus haut sont loin d'avoir atteint toutes les sphères de l'enseignement. Ils n'ont pas été menés jusqu'au bout de leurs conséquences pédagogiques, ce qui n'était probablement pas de la compétence des universitaires qui les ont conduits.

L'empirisme prévaut encore dans de nombreux ouvrages. Il semble à peu près compris que programmer n'est pas résoudre un problème, c'est le faire résoudre par une machine. Les élèves ou les étudiants savent résoudre la plupart des problèmes qu'ils auront à programmer, qu'il s'agisse de trouver quel jour de la semaine tombe un jour donné, ou d'écrire en lettres un montant donné en chiffres, ou de trier une suite de nombres. Or pour

faire faire un travail par un exécutant qui n'a pas été construit de façon spécifique pour ce travail là (les ordinateurs sont des machines universelles à traiter l'information), il faut fournir une méthode à cet exécutant. On est donc confronté au problème suivant : formuler la méthode de résolution d'un problème que nous savons résoudre.

Certains pensent que la difficulté est dans la formalisation de la méthode : nous savons faire, mais nous savons mal dire comment nous faisons. On essaie alors d'expliciter notre propre façon de faire. On manifeste l'enchaînement de nos actions au moyen d'un organigramme, puis on code le résultat dans un langage de programmation : l'analyse informatique part d'une analyse du vécu. Il y a des cas où c'est efficace : s'il s'agit d'échanger les valeurs de deux variables, on regardera comment je m'y prendrai si au cours d'un repas, il se trouve que j'ai un verre de vin rouge, ma voisine un verre de vin blanc, et si nous voulons échanger ces vins sans échanger de verres dans lesquels nous avons déjà bu : on utilise un troisième verre (à noter qu'il faut supposer le troisième verre ébréché, sinon, je ne vois pas pourquoi je ne me contenterai pas de deux transvasements ...). De nombreux ouvrages sont fondés sur cette démarche pédagogique. Toute programmation commence par l'évocation de la façon dont nous opérerions à la main, puis passe à la rédaction de l'organigramme, et enfin à la codification dans un langage approprié. Cette façon de faire laisse de nombreux détails dans l'ombre (notamment les initialisations, la valeur exacte de l'arrêt de boucle). On attend les essais pour les trouver expérimentalement.

Cette démarche est fondamentalement viciée par une hypothèse fausse : nous opérerions nous-même en suivant une méthode. Quand on étale sur une table des cartes portant des numéros, et que l'on demande à un élève de désigner celle qui porte le plus petit numéro, il le fait aisément s'il n'y a pas trop de cartes. Il ne peut pas dire comment il l'a trouvé, sinon par la formule consacrée "ça se voit", et il a raison. Le plus souvent, nous parvenons au résultat soit par une intuition globale, soit en profitant de tous les courts-circuits qui évitent le recours à une méthode systématique fastidieuse, soit par des arrangements locaux et des retouches. Si je demande aux élèves d'ordonner les cartes qui ont été étalées sur la table, ils le feront en mélangeant diverses méthodes de tri, mettant des petites en tête et des grandes en queue, pour s'apercevoir peut-être plus tard qu'ils s'étaient trompés et qu'il y avait plus petit ou plus grand. Ils corrigeront. Pourquoi opéreraient-ils de façon systématique ? Il suffit qu'ils réussissent ...

La démarche que voici se trouve dans plusieurs livres. Il s'agit de chercher le plus petit nombre d'une suite. On dit alors : "comment fais-je moi-même ? Je prends la première carte comme minimum provisoire. Je la compare à la seconde. Si elle est plus petite, elle demeure le minimum provisoire. Si elle est plus grande, alors c'est la seconde qui est le candidat minimum". Le malheur est que personne n'opère de cette façon. Au mieux, on choisit un candidat minimum parce qu'il semble le plus petit de tous, et on s'assure en survolant les cartes que l'on ne s'est pas trompé. Si c'est le cas, on a un nouveau candidat, et on refait la vérification ... On n'opère pas comme le disent les auteurs de ces

ouvrages. En réalité, on a une méthode d'enseignement qui part du programme à écrire, puis le paraphrase en français courant en disant que c'est comme cela que nous faisons. C'est une duperie inacceptable.

On a dit que la difficulté de la programmation venait non de ce que l'élève devait résoudre un problème, mais qu'il devait en résoudre une infinité. On ne lui demande pas de classer ce paquet de dix cartes que j'ai étalées sur la table, mais n'importe quel paquet dans n'importe quel ordre de n'importe quel nombre de cartes. Là encore, je ne crois pas que ce soit la vraie difficulté. Tous les élèves classeront le paquet de dix cartes quel que soit l'ordre initial. Il classeront des paquets de n'importe quel nombre de cartes tant qu'il ne deviendra pas exagérément grand (mais c'est aussi vrai d'un programme : ce qui est écrit pour des suites de quelques centaines, voire quelques milliers de nombres ne vaut plus pour des millions de nombres, ne fut-ce qu'en raison de la taille de la mémoire). Ce n'est pas parce que les élèves savent résoudre tous les problèmes de tri qu'ils savent les faire résoudre par une machine : la variété de situations n'impose pas l'unicité de méthode à l'exécutant humain. La méthode est imposée par la nécessité de commander l'ordinateur : elle demeure même si je veux utiliser mon programme une seule fois sur un jeu bien spécifique de données.

L'enseignement de la programmation se heurte à cette difficulté fondamentale : comment fabriquer une méthode pour résoudre un problème ? On appelle cela "l'algorithmique". Le terme me paraît abusif : l'algorithmique est l'étude des algorithmes, de ce qu'ils permettent de calculer, de la façon dont on peut les composer, les prouver, mesurer leur complexité, ... Ce n'est pas cela qui est en cause, c'est bien moins ambitieux : seulement trouver, face à un problème donné, une méthode qui en viendra à bout. Il faut développer la créativité des élèves ou des étudiants.

Il n'y a pas de réponse universelle. Il n'y a pas un algorithme pour la fabrication d'algorithmes. Il y a des façons de faire. Il y a des méthodes de raisonnement plus adaptées que d'autres : la programmation impérative aussi bien que la programmation récursive se fondent sur le raisonnement par récurrence. Il est important de le savoir, parce que cela fournit un cadre intellectuel. Mais la difficulté demeure : comment trouve-t-on une hypothèse de récurrence qui deviendra le coeur de la procédure récursive, ou l'invariant de boucle pour un programme impératif.

Le problème est encore compliqué par la grande variété de publics en cause. Dans les classes préparatoires scientifiques, on peut supposer le raisonnement scientifique parfaitement maîtrisé, une grande habitude de la résolution de problèmes en mathématiques et en physique. Ce sont des appuis solides. Pour tous les problèmes numériques, qui sont ceux qui importent le plus dans ces classes, l'analyse numérique a déjà fourni les algorithmes de base.

Il en va tout autrement dans un premier cycle universitaire de lettres ou de géographie. Mais même si ces deux publics ont en commun une égale peur des

mathématiques, l'enseignement de la programmation n'y prendra pas la même forme. Le futur géographe est intéressé par les bases de données, la cartographie. Le littéraire voudra faire de l'analyse de textes. Si l'enseignement ne s'appuie pas sur les problèmes proches des étudiants, a-t-il quelque chance de les motiver ? Il ne me paraît pas évident de faire des exercices de cartographie ou d'analyse de textes avec des étudiants suivant quelques heures de programmation

On peut supposer que les étudiants sont capables d'abstraction et d'une certaine formalisation. Peut-on faire la même hypothèse avec les élèves du primaire ou des collèges. Alors quelle informatique pour eux ? Quelle pédagogie s'il faut vraiment pénétrer dans l'univers des algorithmes et de leur programmation ? Ont-ils à être créateurs ? Que peuvent-ils gagner à voir fonctionner des algorithmes qu'ils n'ont pas créés ?

Des réponses ont été apportées. Il importe qu'elles soient connues, analysées avec soin, afin que l'on sache si elles ont bénéficié de contextes locaux favorables, ou si elles sont généralisables. Par dessus tout, il importe que la recherche pédagogique en informatique soit prise au sérieux. Le monde universitaire s'en est jusqu'à présent beaucoup trop tenu à l'écart. On ne peut pas continuer avec une université où s'élaborent des théories et des écoles où se pratique l'enseignement, sans des échanges nombreux. Il faut que les universitaires se préoccupent de pédagogie. Il faut que les enseignants puissent se tenir au courant des résultats de la recherche.

C'est le rôle d'un colloque scientifique sur la didactique de l'informatique que de mettre en contact chercheurs et enseignants. Tout ne sera pas fait en une seule rencontre. Il faut espérer qu'elle aura suffisamment de succès pour que se crée un courant qui bénéficiera à la recherche en lui faisant connaître les problèmes vécus par les enseignants, et à l'enseignement en mettant à sa disposition les résultats de la recherche.

BIBLIOGRAPHIE.

[NAU66] P. NAUR. *Proof of algorithms by general snapshots*. BIT 6, 4, 1966, pp. 310-316.

[FLO67] R.W. FLOYD. *Assigning meaning to programs*. Proceedings of symposia in applied mathematics, vol 19, american math. society, 1967, pp. 19-32.

[DIJ68] E.W. DIJKSTRA. *GOTO statements considered as harmful*. Comm. ACM 11, 3, mars 1968, pp. 147-148.

[KNU68] D. KNUTH. *The art of computer programming*, vol 1, Addison Wesley, 1968.

[NAU69] P. NAUR. *Programming by action clusters*. BIT 9, 3, 1969, pp. 250-258.

[SEV70] *L'enseignement de l'informatique à l'école secondaire*. Colloque IFIP - OCDE, Sèvres, 1970. Publications de l'OCDE, Paris 1971.

[DDH72] O.J.DAHL, E.W. DIJKSTRA, C.A.R. HOARE. *Structured programming*. Academic Press, Londres 1972.

[MON73] *The high cost of software Proc*. of a symposium SRI Menlo Park, 1973, Editeur B.W. Boehm.

[WAR73] J.D. WARNIER. *Les procédures de traitement et leurs données*. Editions d'organisation, Paris 1973.

[JAC75] M.A. JACKSON. *Principles of program design*. Academic Press, Londres, 1975.

[DIJ76] E.W. DIJKSTRA. *A discipline of programming*. Prentice Hall 1976.

[ARS77] J. ARSAC. *La construction de programmes structurés*. Dunod, Paris, 1977.

[SIM79] J. C. SIMON. *L'éducation et l'informatisation de la société*. La documentation française, Paris, 1980.

[ARS80] J. ARSAC. *Premières leçons de programmation*. Cedic, Paris, 1980.

[GRI81] D. GRIES. *The science of programming*. Springer Verlag. Berlin, 1981.

[DUC84] DUCRIN. *Programmation*, Paris, Dunod, 1984.