Meta
Journal des traducteurs
Translators' Journal

META

L’enseignement de l’informatique aux traducteurs

Caroline de Schaetzen

Volume 39, Number 1, mars 1994

La traduction et l’interprétation dans la Belgique multilingue

URI: https://id.erudit.org/iderudit/004039ar
DOI: https://doi.org/10.7202/004039ar

See table of contents

Publisher(s)

Les Presses de l'Université de Montréal

ISSN

0026-0452 (print)
1492-1421 (digital)

Explore this journal

Cite this article

de Schaetzen, C. (1994). L’enseignement de l’informatique aux traducteurs. *Meta*, *39*(1), 57–68. https://doi.org/10.7202/004039ar

L'ENSEIGNEMENT DE L'INFORMATIQUE AUX TRADUCTEURS

CAROLINE DE SCHAETZEN
Institut Libre Marie Haps, Bruxelles, Belgique

Faut-il enseigner aux étudiants en traduction d'aujourd'hui des outils informatiques dont on connaît la rapide obsolescence? Les traducteurs et traductologues sont-ils en majorité des utilisateurs de programmes faits par d'autres ou doivent-ils en être créateurs? L'étudiant doit-il considérer l'ordinateur comme une boîte noire, comme le fait l'automobiliste pour son véhicule? Quelle définition donner d'une culture générale en traductique? Quelle éducation donner pour former l'esprit critique sur les outils d'aide à la traduction existants, dans un monde professionnel en pleine informatisation?

C'est par leur pratique qu'en 1991 les professeurs d'informatique répondent à ces questions pédagogiques; les établissements et facultés d'enseignement de la traduction initient aux outils d'aide à la traduction du marché:

- un progiciel standard de traitement de texte;
- la consultation d'une ou plusieurs banques de données terminologiques;
- l'utilisation et l'alimentation d'une base de termes (souvent interne à l'établissement);
- un gestionnaire de glossaires ou d'un système de gestion de bases de données;
- un logiciel d'exploitation.

Pour l'an 2000, cette formation n'est plus suffisante. Quelques instituts ou facultés enseignent déjà la programmation, à l'aide d'un langage spécialisé dans les applications linguistiques; dans ce cas, les exercices et travaux des étudiants portent évidemment sur la traduction automatique. L'Institut Marie Haps de Bruxelles organise quant à lui depuis six années, pour ses étudiants de troisième année en traduction et en interprétation, un cours de programmation dans le langage d'un progiciel. Les premières années, le langage choisi était celui de DBASEIII puis de DBASEIII PLUS; depuis 3 ans, c'est celui des macrocommandes du traitement de texte WORD5. Nous présentons ici le bilan de cette activité pédagogique.

Le traitement de texte est le compagnon de travail par excellence du traducteur professionnel (dont le métier restera *manuel* à court terme), comme de la secrétaire et de l'écrivain: actuellement, tous les travaux des traducteurs tournent autour des manipulations de texte. Associé à un gestionnaire de glossaires, un logiciel de traitement de texte de haut niveau peut pratiquement tout faire pour un traducteur, sauf sa comptabilité et l'automatisation complète de sa traduction. L'utilité de l'informatisation de ces manipulations textuelles, surtout celles de bas niveau (publipostage d'offres de services, de factures, classement du courrier, de la documentation) est immédiate, notamment pour les traducteurs indépendants. Actuellement, et aussi paradoxal que cela puisse paraître, la programmation de la traduction proprement dite (même à l'aide d'un langage dédié à cette application) ne nous semble pas encore indiquée pour les étudiants en traduction, qui suivent un enseignement essentiellement technique. Elle relève encore, au stade actuel de l'informatisation de la traduction, de la recherche pure: elle est menée par de grandes équipes

interdisciplinaires et nécessite, plus qu'une connaissance de plusieurs langues, un bagage solide en linguistique, en logique, en épistémologie cognitive et en programmation. Aussi, ce sont les facultés de linguistique et d'informatique qui mènent la plupart des grands projets de traduction automatique. Il va de soi que le futur traducteur doit par contre disposer de connaissances suffisantes pour juger de la qualité des logiciels de traduction automatique existants ou à venir...

Il n'existe dans le commerce aucun programme *autour* des logiciels de traitement de texte. Lorsqu'ils feront leur apparition, il y a gros à parier que leur prix (comme celui des feuilles de style pour les logiciels de publication assistée, par exemple) sera prohibitif, que leurs fonctions seront adaptées aux besoins de chaque traducteur et à l'évolution de ces besoins.

Les traducteurs ont en outre besoin d'une série de petits programmes qui ne servent que temporairement, voire une seule fois (par exemple, recopie d'une entité structurée d'une certaine manière d'un document à l'autre, remplacement conditionnel d'une chaîne de caractères dans une traduction précise, mise en page particulière, etc.); ces programmes doivent donc pouvoir être conçus et rédigés très rapidement.

Comme tous les progiciels standard permettant la création et le stockage de macrocommandes, les traitements de texte standard sont pourvus d'un enregistreur de commandes. Le recours à ce dispositif accélère grandement le déroulement des opérations. En effet, l'appel groupé de ces commandes ne nécessite que la pression d'une combinaison de touches. De plus, les commandes sont compilées. Aussi lors de l'exécution de la chaîne des commandes enregistrées, chacune d'elles se déroule-t-elle à une vitesse beaucoup plus grande qu'en mode interactif. C'est une amélioration par rapport aux progiciels plus anciens de types DBASE, où, les commandes étant interprétées, elles s'exécutent lentement. Peu de traducteurs utilisent le mode *macro* d'exécution des commandes[1]...

Le mode *macro* recèle en outre un langage de programmation véritable et complet, qui permet d'enregistrer dans les programmes écrits avec lui des séquences de commandes du logiciel lui-même.

En langage macro et moyennant des connaissances en programmation très élémentaires, des applications puissantes peuvent être écrites quatre fois plus rapidement qu'avec les instructions des langages dits de *haut* (il faudrait à présent dire *bas*) niveau comme le BASIC ou le PASCAL. Les commandes du logiciel de traitement incluses dans les programmes sont en effet beaucoup plus puissantes qu'une instruction en langage de programmation classique. Ainsi, un programme de création et d'inversion d'index de dictionnaire écrit dans le langage macro de WORD5 comporte une page de commandes et d'instructions de structuration, pour une vingtaine d'instructions dans un langage de type PASCAL.

Si l'apprentissage d'un logiciel à l'aide d'un manuel d'utilisateur est aisé, celle de la programmation, même par macrocommandes, requiert un enseignement en bonne et due forme, notamment des exercices nombreux. La didactique de la programmation est ardue pour les esprits dits *littéraires* qui peuplent encore (trop?) les facultés et instituts d'enseignement de la traduction: les non-scientifiques et les non-techniciens répugnent au formalisme; la gymnastique mentale consistant à modéliser un problème donné sous forme d'objet du monde en une équation formalisable (équation algébrique, par exemple) leur fait défaut. Pire: la simple vue d'un symbole logique et *a fortiori* mathématique, bloque leur esprit... À cet égard, les règles de syntaxe et de sémantique rebutantes des langages de *haut* niveau complique l'étude de ces langages, par contraste avec la légèreté des contraintes syntaxiques des langages macros. Les règles de transcription des touches, de l'abréviation des libellés de menus et des options peuvent même être contournées par les programmeurs débutants: dans un programme simple, seules les instructions de structures et d'affectation doivent être écrites par lui, le reste peut être enregistré (par opposition au

DBASE, par exemple, où l'écriture de toutes les commandes est requise). De plus, l'emploi du traitement de texte lui-même, c'est-à-dire le passage entre les actions complètes qu'on veut faire et l'enchaînement des commandes à donner pour y arriver, prépare déjà le futur traducteur à la programmation : maîtriser un progiciel, c'est avoir perçu les tâches qu'il rend possibles et avoir acquis les tours de mains sous-jacents à l'exécution de chacune de ces tâches ; or, chaque tour de main est un petit algorithme mental dont l'utilisateur devient l'exécutant.

Un problème de programmation change de caractère selon le type de langage utilisé : en programmation impérative (langages BASIC, PASCAL), on dit comment un résultat est calculé, en programmation fonctionnelle (PROLOG), ce qui est *vrai* dans la solution d'un problème. La programmation impérative est plus proche, par sa démarche séquentielle, des traitements humains d'opérations que la programmation fonctionnelle, reposant sur la récursion. De plus, la programmation impérative bénéficie d'une longue tradition didactique, notamment d'outils pédagogiques nombreux (générateurs d'organigrammes, de plans, de schémas, pseudo-langages informatisés). Or, par opposition aux langages fonctionnels utilisés en linguistique informatique, les langages macros sont impératifs.

Ces langages sont également modulaires (une macro peut s'appeler elle-même) ; ils permettent donc la programmation par assemblage de routines et facilitent l'échange de programmes.

Pour les macrocommandes qui ne fonctionnent pas, le déroulement pas à pas peut être demandé. Les messages d'erreurs et les arrêts lors de l'exécution de programmes comportant des erreurs facilitent également à l'apprenti programmeur le *débogage* de ses programmes.

La didactique de l'algorithmique constitue, on le verra, le *noyau dur* de l'enseignement de la programmation. Traditionnellement, la recherche d'algorithmes se fait donc à l'aide de pseudo-langages, dont plusieurs disposent d'un interpréteur. Des pseudo-langages didactiques peuvent être très facilement conçus avec et pour les langages macros, grâce à la convivialité de ces langages.

À l'instar de la conduite d'une voiture, la programmation est désormais un savoir-faire[2] (et un jeu !), qui relève déjà de la culture générale.

Le programmeur se distingue par la rigueur de sa pensée et la précision de son expression. Il y a plus. Savoir programmer, c'est-à-dire passer «d'un problème sur des objets du monde au texte d'un programme exécutable par un langage informatique donné» (C. Pair), c'est être capable de formaliser sa méthode de résolution d'un problème. Cette *science de la méthode* est donc une métaconnaissance : elle ne décrit pas les procédures à utiliser, mais guide l'utilisateur dans la recherche de stratégies de résolution de problèmes d'un certain type[3].

Lorsqu'elles sont maîtrisées, la facilité d'utilisation des heuristiques et de décomposition des problèmes, la conscience de la pluralité des solutions possibles à un problème, la rationnalité de l'activité par rapport à une fin, la gestion univoque des moyens, la logique sans ambiguïté, la vue de la recherche des erreurs comme activité positive (*débogage* des programmes) se transfèrent souvent[4] à d'autres volets du savoir-faire du traducteur. Ces connaissances et métaconnaissances lui sont précieuses : sa profession requiert la précision de la pensée et de l'expression ainsi que l'art d'apprendre rapidement (nouvelles langues ou nouveaux aspects des langues à étudier, nouvelles disciplines, nouvelles techniques de recherche documentaire, nouveaux outils d'aide à la traduction).

Chaque type de progiciel et son langage macro mobilisent en outre des facultés cognitives différentes : le monde *historique* des commandes séquentielles du DOS, l'optique relationnelle des gestionnaires de bases de données, la vue *spatiale* des logiciels de traitement de texte, dans lesquels le curseur peut se déplacer n'importe où à l'écran. À terme,

l'extension de la programmation avec traitement de texte doit encore susciter, comme l'a fait l'automatisation des applications autour des gestionnaires de bases de données, des *mises à plat* de l'expertise dans les manipulations de texte elles-mêmes.

Certains pas qualitatifs[5] peuvent être chronologiquement distingués dans le processus d'apprentissage d'un langage de programmation, que ce soit celui des macrocommandes ou d'un langage classique :

1. capacité d'inférer, grâce à l'utilisation de programmes, ce qu'un ordinateur peut prendre en charge comme activités (niveau de l'utilisateur) ;
2. connaissance de la syntaxe et de la sémantique des principales instructions d'un langage de programmation ; capacité de réaliser des programmes courts sur le mode de ceux qui ont été montrés et utilisés ;
3. capacité de raisonner en termes d'unités de haut niveau (connaître des séquences d'instructions réalisant certains sous-objectifs fréquents, comme compter sélectivement des entités d'un document) et capacité d'écrire des programmes longs mais peu conviviaux (niveau du programmeur débutant) ;
4. capacité d'écrire des programmes complexes et structurés, orientés vers les utilisateurs (prise en compte de l'éventail de leurs erreurs et souhaits possibles, par exemple), avec distance par rapport au code, concentration sur les phases et les processus de résolution du problème. Faculté de commenter ses programmes et d'en faire l'analyse préalable (niveau du programmeur, du créateur de logiciels).

Dans le cadre des études de traduction, les tâches d'apprentissage ne dépasseront pas, bien sûr, le stade 3 : l'étape suivante nécessite trop de temps et d'exercices et est réservée aux programmeurs professionnels. Cependant, même les programmes écrits au stade 2 seront directement utiles au traducteur professionnel.

L'apprentissage de la programmation pose certaines difficultés. Un algorithme est fait d'actions modifiant des situations. L'art de modéliser est difficile : il n'existe pas d'algorithme pour la création d'algorithmes (comment trouver l'invariant d'une boucle ou l'hypothèse de récurrence ?).

La programmation fait appel à un ensemble de notions en interaction : un traitement de glossaires, donc de listes, implique les notions de variables, d'itérations, de fonctions opérant sur des sous-ensembles de ces listes, des tests, des entrées / sorties... Les débutants ont dès lors du mal à penser à tout, notamment aux détails. Les fautes de distraction les plus fréquentes sont les suivantes :

- erreurs dans les commandes du curseurs ;
- omission de la fermeture de boucles ;
- oubli d'affichage des résultats ;
- omission d'initialisation des variables ;
- fautes d'orthographe dans les libellés de touches...

Souvent, la recherche empirique de l'algorithmique repose sur l'observation des opérations effectuées par l'être humain pour la solution du problème posé. Après cette observation, l'enchaînement des opérations est noté sur un organigramme, puis les opérations sont codées dans le langage de programmation enseigné. Cette transposition se heurte à deux difficultés :

- son caratère incomplet (il manque l'initialisation et la valeur exacte de l'arrêt de boucle, par exemple), la situation d'exécution mentale se faisant au niveau du détail et non des grandes fonctions à remplir par le programme ;

- le fait que l'être humain procède souvent intuitivement (par des associations, par d'autres courts-circuits évitant le recours à une méthode systématique fastidieuse) ou par retouches et arrangements locaux.

Au tout début de l'apprentissage de la programmation, l'anthropomorphisme est donc fréquent :

- une procédure humaine est traduite littéralement dans le langage de programmation (emploi de la valeur *VRAI* des variables logiques au lieu de la vérification de la valeur de vérité) ;
- des informations implicites sont omises. Les règles de fonctionnement de la présupposition dans le langage naturel sont notamment un modèle traître pour l'écriture de conditions en programmation. Selon elles, lorsque la condition A a été traitée, les autres instructions s'appliquent à la situation non-A et, partant, il n'est pas nécessaire de spécifier cette condition non-A dans le texte du programme ; inversement, tant que le traitement de la condition A n'est pas achevé, les instructions s'appliquent au cas A. D'où la difficulté qu'éprouve le débutant à exprimer exactement les conditions des itérations, surtout si elles doivent être énoncées de manière négative.

L'écriture de programmes itératifs ou récursifs implique en outre la maîtrise de la dialectique, neuve pour l'apprenant, entre exécution (dynamique) d'actions et succession (statique) de situations ou de relations.

Enfin, les règles du jeu peuvent changer à l'intérieur de la programmation elle-même : dans les boucles, le contenu de la variable est fonction de l'exécution et il est donc choisi par le programme, non par l'utilisateur, ce qui contredit le passé cognitif immédiat de l'apprenant.

Les stratégies générales de résolution de problèmes (inférence, formalisation, généralisation d'hypothèses) de l'apprenant jouent un rôle non négligeable dans l'apprentissage. Un manque de pratique en formalisation entraînera, par exemple, les erreurs suivantes :

- confusion entre les connaissances sur le monde et leur codification ;
- ignorance de ce qu'est une fonction ;
- méconnaissance profonde du concept de variable (confusion entre nom et valeur d'une variable, entre affectation et demande à l'utilisateur du programme d'affecter une variable, non-emploi de variables intermédiaires) ;
- confusion entre les nombres cardinaux et ordinaux dans la fonction *longueur de chaîne* ;
- confusion entre ce qui est conventionnel (aspects de la syntaxe, par exemple) et ce qui ne l'est pas.

Une difficulté de catégorisation entraînera des difficultés à distinguer l'essentiel de l'accessoire, à hiérarchiser les éléments d'un problème (erreurs de séquence, telle l'insertion d'instructions de portée trop générale dans une boucle).

Au début, les conceptions que se font les étudiants de la *boîte noire* du dispositif informatique dans son ensemble peuvent elles aussi interférer avec l'apprentissage. S'il confond, par exemple, la *machine notionnelle* du langage de programmation et celle du système d'exploitation, l'étudiant ne fera pas la distinction entre l'édition et l'exécution d'une macro (il tentera de lancer une macrocommande sans être sorti du mode *enregistrement* de macro, affectera des variables pendant l'édition du programme).

Des attitudes de l'apprenant peuvent également nuire à l'apprentissage: certains étudiants ne révisent pas leurs prémisses en cas d'échecs, même répétés, de leurs programmes. Ils ne font pas de *débogage actif*: ils n'utilisent pas les messages d'erreur, ne testent pas leur programme, ne formulent pas d'hypothèse s'il fonctionne mal... Ce comportement peut être induit soit par l'image négative qu'ont ces étudiants de leur propre compétence, soit par l'importance que revêtent à leurs yeux les connaissances nouvellement acquises; cette importance serait telle qu'elle masquerait leur démarche intellectuelle elle-même... Par pur ritualisme, d'autres étudiants utilisent systématiquement une ou deux instructions bien précises.

L'adage veut que les performances des apprenants soient directement liées à la compétence du maître. Quel est à cet égard le degré de directivité de l'enseignant? Néglige-t-il, par exemple, de partir du raisonnement de l'apprenant en cas d'erreur, alors que des débutants n'ont pas forcément le même cheminement vers une solution que lui? Quel accent met-il sur la syntaxe? Individualise-t-il son enseignement, lui qui le premier domine un outil permettant la diversification et la personnalisation des apprentissages?

Le besoin de séparer les rôles qu'éprouvent certains responsables de l'enseignement est pernicieux: ces professeurs compliquent inconsciemment leur enseignement, pour retarder l'accès au savoir des étudiants, confirmant et valorisant par là même leur propre expertise et leur statut de détenteur du savoir:

- ils omettent de démystifier la programmation en donnant des points de comparaison familiers;
- ils négligent d'avertir les étudiants du caractère incontournable des obstacles;
- ils n'exposent pas le raisonnement qu'ils ont effectué eux-mêmes pour solutionner un problème de programmation, présentant d'emblée la solution complète;
- ils ne décomposent pas suffisamment ce raisonnement (la construction d'un algorithme n'est pas la paraphrase en français courant d'un programme dans un langage macro particulier)...

Tout aussi inconsciemment, les étudiants essayent de surmonter cette barrière: une série d'entre eux persistent à prendre des notes malgré la mise à leur disposition de manuels de programmation complets; or plus ils prennent des notes, plus ils postposent la réflexion, surtout en l'absence d'execices cotés[6].

L'environnement pédagogique global peut également présenter des défauts:

- les modalités d'accès aux machines peuvent être restrictives, ce qui réduit le nombre d'exercices;
- certaines activités pédagogiques manquent de continuité: s'il est séparé, dans le temps et dans l'espace, des cours de traduction, le cours d'informatique risque de se voir conférer par les étudiants un statut comparable à la gymnastique ou à la natation au secondaire, c'est-à-dire de ne pas être perçu comme une démarche intellectuelle insérée dans leur discipline. Cette parcellisation de l'enseignement empêchera également la micro-expertise acquise par la programmation de se transférer à d'autres domaines d'activité de l'étudiant.

À matières neuves, pédagogie renouvelée. Or la didactique de l'informatique n'est pas simple: à quel moment de l'enseignement faut-il introduire explicitement telle méthode? Dans quel cas les méthodes sont-elles centrées sur l'utilisation efficace de connaissances acquises et sur l'acquisition même d'un contenu? Comment réduire les disparités, sachant que, dans une population normale, la durée d'apprentissage varie de un à quatre?

D'une manière générale, l'apprentissage doit être parcellisé à l'extrême, comme il l'est, par exemple, dans les tutoriels.

L'enseignement du langage macro doit être distingué clairement de celui de l'algorithmique et des structures de programmation. Lorsque le professeur a choisi d'utiliser un pseudo-langage pour enseigner l'algorithmique, il ne peut, par exemple, l'exposer avant des rudiments du langage de programmation, comme cela se fait encore trop souvent, sans quoi les apprenants ne sauront quand s'arrêter d'analyser, ni même ce qu'est un algorithme.

L'ordre des matières à exposer pour l'enseignement du langage macro pourrait être le suivant :

1. séquentialité ;
2. variables et affectation ;
3. boucles simples ;
4. boucles avec compteur ;
5. boucles imbriquées ;
6. règles du codage des touches et des libellés de menus ;
7. stratégies de traitement de texte.

Lors d'un exposé d'une structure de programmation, les instructions complexes gagnent à être décomposées :

- la stucture TANT QUE, qui est en fait une condition («s'il n'y en a plus, fini !»), insérée à la fin d'une boucle de répétition ;
- la demande d'affectation par l'utilisateur du programme ;
- les fonctions ;
- la formule *pour i allant de 1 à N*.

L'illustration de chaque structure par la modification progressive d'un exemple unique permet à l'étudiant de se concentrer sur la compréhension de la nouvelle structure et lui en fait appréhender immédiatement la spécificité. Un programme illustratif peut ainsi être complété petit à petit et lancé à chaque étape de l'exposé. Dans une addition ou concaténation unique de deux chaînes de caractères, par exemple, dont le résultat est affiché, le professeur peut :

1. remplacer un chiffre ou une chaîne par une variable ;
2. rajouter une condition ;
3. rajouter une alternative à cette condition ;
4. faire affecter la variable par l'utilisateur de la macro ;
5. rajouter une condition plus globale, etc.

Le ralentissement de l'exécution d'un programme par le mode exécution pas à pas illustre bien le fonctionnement de certaines structures complexes (boucles, variables).

La communication des activités de l'ordinateur pendant l'exécution du programme est tout aussi instructive :

- écriture, à la fin du corps des boucles, de l'état des variables d'un programme centré sur une structure itérative, car elle permet de distinguer les variables qui gardent leur valeur pendant toute la durée d'exécution de celles dont la valeur change pendant le parcours de la boucle ;
- affichage continu du contenu des variables ;
- affichage des résultats intermédiaires ;
- affichage d'un pointeur montrant l'exécution de chaque instruction au cours du déroulement du programme.

Les tâches d'apprentissage peuvent être subdivisées, elle aussi, l'écriture de programmes, complexes ou simples, n'étant alors que l'ultime d'entre elles :

- la lecture et l'emploi de programmes existants permettent d'inférer des faits et d'apprendre par imitation ;
- le complément et la modification de programmes peuvent être tout aussi formateurs que l'écriture d'un programme simple, dont les erreurs de syntaxe risquent d'absorber une part non négligeable de l'énergie de l'apprenant ;
- les exercices sur papier permettent de lutter contre l'anthropomorphisme ou d'enseigner des structures de programmation avant la syntaxe d'un langage macro précis, par exemple : CH étant une chaîne, écrire la condition disant si CH est vide, est «oui» ou «non», commence par un «b» (enseignement de la condition) ; mettre au féminin des substantifs dont on choisit le nombre, inverser une chaîne de caractères (boucle avec compteur) ; conception de l'algorithme du café, indentation des structures d'organisation hiérarchiques administratives d'une province belge (imbrication d'instructions). Il est cependant important de ne pas commencer l'enseignement par ces exercices, trop éloignés des applications de traduction ;
- parmi les premiers problèmes choisis par l'enseignant, un certain nombre peut relever de la catégorie pour laquelle il n'existe pas de solution humaine (par exemple : un programme affichant l'année de naissance de l'utilisateur lorsque celui-ci a tapé son numéro d'identification INSEE). Par ce choix, les élèves se heurteront aux spécificités du dispositif lorsque celles-ci contrediront leurs représentations de ce dispositif et ce choix les contraindra donc à un codage efficace.

Voici un exemple de structure de cours qui sépare efficacement l'enseignement de l'algorithmique de celui d'un langage de programmation tout en les faisant alterner :

1. rappel ou présentation de la notion du langage à utiliser dans la séance d'exercices ;
2. distribution d'un énoncé et présentation du fonctionnement d'une solution (fourniture d'un programme existant rédigé dans un pseudo-langage et qui se commente lui-même) ;
3. construction ascendante de la solution sur papier (pour que l'étudiant ne tape pas n'importe quoi !) ou sur écran. Cette construction peut consister à remplacer une instruction par une autre ou par plusieurs autres ;
4. communication par l'enseignant de la syntaxe de la commande dans le langage macro et dactylographie par les étudiants de la commande dans un autre programme parallèle, rédigé dans le langage macro ;
5. recherche par les étudiants d'un complément de solution à domicile (avec remise au professeur, sur disquette, de l'algorithme, du détail de la conception et du programme).

L'utilité des méthodes d'algorithmique n'apparaissant que dans les premiers projets réels et consistants, il faut éviter de les présenter *a priori* et à partir d'exercices trop simples, sous peine de donner aux étudiants l'impression que ces méthodes ne sont que l'émanation de déformations professionnelles tatillonnes. D'autant que, pour le débutant, le jugement de la machine est plus important que celui du professeur (dès que *ça tourne*, il cesse d'écouter les explications !).

D'où l'intérêt de la pédagogie par projet, en l'occurrence, la programmation d'une application de taille et d'utilité réelles (idéalement, une commande de l'extérieur), dans le domaine de la traduction. À la suite d'un stage des étudiants dans un bureau ou service de traduction, le professeur peut, par exemple, faire dresser la liste de toutes les opérations

effectuées pour une prestation précise (demande de devis de traduction, par exemple) ; les cas divergents (traduction sous-traitée à un collègue) seront précisés dans cet inventaire. L'avantage subsidiaire de ce pointage sera la confrontation des solutions trouvées dans les différents endroits de stage des étudiants. L'établissement d'enseignement peut aussi organiser, dans la salle d'ordinateurs, un cours de traduction pour lequel des programmes seraient écrits par les étudiants eux-mêmes. Le projet peut être confié à tout un groupe, ce qui a l'avantage :

- d'illustrer directement le caractère indispensable des procédures, comme autant de *briques* pour l'édification d'un programme complet ;
- de faire faire à chacun quelque chose de différent, tout en gravant dans tous les esprits les contraintes d'intégration des différents modules à l'ensemble (la première analyse et d'intégration des modules doivent donc être menées en commun) ;
- de provoquer une explicitation des stratégies, donc une réflexion des étudiants sur leurs propres méthodes ;
- de montrer la multiplicité possible des approches d'un même problème.

Pour prouver l'intérêt du dossier (comprenant données, résultats, liste de variables ainsi que leur contenu et la manière dont les résultats sont obtenus des données), qui rend le programme communicable, le départ des groupes peut être simulé ou organisé (ledit groupe étant alors contraint de remettre le dossier du programme en l'état à son successeur).

Il est à noter que, pour ces travaux de groupes, un module de trois étudiants est conseillé : à deux, les étudiants se créent un jargon et, à quatre, les problèmes d'organisation inhérents au travail en groupe font leur apparition. Le changement de partenaires est également préférable. Enfin, l'enseignement intensif a le mérite de modifier les techniques utilisées, jusqu'à créer de vrais réflexes, s'il dure une semaine à temps plein, et à créer une certaine pression au sein du groupe.

La découverte d'un algorithme par l'étudiant peut être, sinon induite, du moins facilitée. Une philosophie générale peut être proposée pour cette quête :

- trouver un algorithme, c'est définir les situations initiales et les finalités (Que sait-on ? Que cherche-t-on ?) ;
- les techniques de programmation descendante et ascendante sérient les difficultés (*cf. infra*) ;
- si le but ne peut être atteint en une fois, on détermine une situation intermédiaire qui peut être atteinte à partir des données, et dont on sait qu'elle rapproche déjà du but.

Des activités peuvent être conseillées :

- un travail de réflexion sur le dictionnaire des données (décrire le sens et le rôle de chacune, vérifier qu'il n'y a qu'une donnée par information utile et qu'une information manipulée par donnée), notamment pour faciliter la détermination de conditions posant des problèmes ;
- une liste de *chose à essayer* (comparer le contenu des chaînes, déplacer le curseur, compter certaines entités) ;
- la lecture de programmes correspondant à un problème similaire à celui qui est posé.

Des règles peuvent être recommandées :

- l'analyse soigneuse du problème ;
- la spécification des objectifs et des concepts utilisés à tous les niveaux et l'identification de chaque concept par un nom qui lui est propre ;

- la distinction claire de ce qui est déjà fait et de ce qui reste à faire ;
- le principe de *moindre compromission* (une décision ne doit être prise que s'il est impossible de faire autrement).

Les difficultés qu'entraîne l'apprentissage de la programmation devraient interdire tout dogmatisme aux responsables de l'enseignement. Le problème *dans le monde* peut être résolu dans son domaine, puis sa solution traduite dans le langage macro ; mais les données et les questions du problème peuvent tout aussi bien être interprétées dans ce langage macro et le nouveau problème traité avec cet outil. De la même manière, les professionnels procèdent à une combinaison des méthodes descendante et ascendante pour trouver l'algorithme d'un problème. Pourquoi trop de professeurs d'informatique imposent-ils alors la seule méthode descendante[7] ? Si elle a l'avantage de fragmenter les problèmes et d'orienter le traitement en fonction des données connues, elle implique d'envisager dès le départ le problème dans sa totalité, et la représentation n'est pas indépendante d'une anticipation sur le type de traitement possible. En fait, la démarche est plus souvent ascendante au stade de l'initiation.

Les outils de conception de programmes (dictionnaires de données, plans, schémas) posent problème si le pseudo-langage n'est pas assez riche dans ses structures de données, mais, au-delà des querelles d'école sur leurs mérites respectifs, ils favorisent tous les activités cognitives en jeu dans la mise en œuvre de stratégies de résolution de problèmes. Un même formalisme peut être utilisé à plusieurs niveaux : stratégie, tactiques locales, implémentation. En outre, si l'insistance doit être mise sur la rigueur du raisonnement, pour l'utilisation d'un formalisme algorithmique, la liberté d'écriture doit être accordée en raison des résistances suscitées par les formalismes.

Les travaux pratiques que permet la multiplication des micro-ordinateurs dans les établissements d'enseignement sont capitaux : si la programmation s'enseigne au cours, elle s'assimile lors des travaux dirigés.

Contrairement à certaines idées reçues, les tests de programmes sur machine ne sont pas antipédagogiques. Quel programmeur professionnel ne recourt jamais à la mise au point par essais et erreurs, c'est-à-dire à l'assistance du compilateur du programme, ne fût-ce que s'il est fatigué ? En outre, la constatation *de visu* d'un succès est motivante pour le débutant en programmation. Les tests n'empêchent nullement le professeur d'imposer, avant le lancement d'un programme, une vérification de la cohérence de l'ensemble de ses instructions et une simulation mentale rapide de son exécution.

Trop souvent, l'enseignement supérieur fait *fi* des *grosses ficelles* pédagogiques. Celles-ci constituent pourtant des recettes éprouvées (elles ont été élaborées et testées longuement au secondaire et au primaire). Avant de donner la solution, il est, par exemple, souvent utile d'orienter l'apprenant vers des solutions plus simplistes (itérations au lieu d'une récursion, par exemple) mais moins efficaces, ce qui lui permet de mieux comprendre le problème posé.

Les métaphores facilitent la compréhension des propriétés d'un concept : l'analogie, la transposition sont une des étapes de la compréhension ; elles rapprochent du réseau sémantique de l'apprenant et démystifient la programmation, par leurs références à des points de repères familiers. La décomposition peut, par exemple, être illustrée par l'exemple fameux des recettes de cuisine ou l'algorithme du café, les mouvements au sein de répertoires, celui de l'armoire et des tiroirs...

Les représentations visuelles, fixes ou animées, ne sont pas non plus indignes d'un professeur universitaire :

- lors de l'écriture du programme, la disposition en retraits successifs des lignes d'instructions montre leurs niveaux d'imbrication ;

- un branchement sur un organigramme peut être illustré par le film d'un automobiliste envisageant soit de changer un pneu crevé soit de faire de l'auto-stop vers un garage.

Comme dans toute activité d'apprentissage, les étudiants dont les prestations sont les plus faibles sont ceux qui s'en rendent le moins compte... L'appréciation des performances est donc une étape à part entière de l'enseignement lui-même.

Les corrections d'exercices écrits permettent au responsable du cours de mesurer l'écart entre les objectifs pédagogiques et les réalisations des apprenants, ainsi que de recenser de manière précise les blocages et les erreurs.

Par les évaluations de groupe, l'étudiant peut s'auto-évaluer par rapport aux autres membres du groupe. Exemple : des réunions de bilan d'un quart d'heure, visant à déterminer quel était le travail le plus ou le moins satisfaisant et pourquoi ; ces comptes rendus permettent l'expression des inquiétudes et leur interprétation positive (comme prise de conscience de l'importance des questions soulevées, non comme manifestation d'une impuissance, dont les autres seraient témoins et juges).

Une grille d'évaluation uniformise les corrections et facilite leur commentaire aux apprenants. Il est important que les critiques des prestations soient les plus concrètes possibles (puisqu'il ne peut être objectif, le professeur se doit d'être équitable !).

La cotation des exercices sur 5, au lieu de 10 ou 20, empêche la contamination de la fonction pédagogique de l'évaluation par la fonction institutionnelle de contrôle (les élèves voulant surtout *réussir le test*).

Traduction, didactique et informatique s'éclairent mutuellement. Programmer, c'est enseigner à l'ordinateur à faire quelque chose que nous savons faire, mais c'est aussi traduire des instructions dans un autre langage.

Les outils d'aide à la didactique de la programmation le montrent, l'informatique change la didactique, en commençant par celle de l'informatique.

La didactique change elle aussi l'informatique : toutes les matières d'examen font l'objet d'une ritualisation, d'une réification. À cet égard, de même que l'épistémologie empêche les sciences de fonctionner lorsqu'elle se veut prescriptive, la didactique contrevient à l'efficacité d'un enseignement si ses prétentions sont normatives. La pédagogie par projet, cette tentative de revenir à la formation sur le tas en la simulant, en est le signe... Aussi les recommandations données ci-dessus ne doivent-elles être comprises que comme la relation d'une expérience, dont l'intérêt est purement illustratif...

Notes

1. Trop d'entre eux méconnaissent en outre bon nombre de fonctions de traitement de texte qui leur seraient précieuses.
2. C'est malheureusement comme telle que la programmation est un peu méprisée dans l'enseignement universitaire.
3. «La révolution informatique est une révolution dans le mode de la pensée et dans son expression. L'essence de ce changement est l'émergence de ce qu'il conviendraient d'appeler une épistémologie des processus, c'est-à-dire l'étude des structures de la connaissance d'un point de vue opérationnel, contrairement à l'approche déclarative communément utilisée en mathématiques.» (Abelson et Sussman, *Structure et interprétation des programmes informatiques*, InterEditions, Paris)
4. Le transfert des compétences à d'autres domaines n'est cependant pas automatique : comme on le verra plus loin, la programmation interagit avec les connaissances antérieures de l'apprenant et avec l'environnement pédagogique.
5. Selon le modèle de l'apprentissage dressé par Piaget en parallèle de celui de l'évolution des sciences, où les interactions avec l'environnement alimentent une certaine compréhension des choses, jusqu'à ce que les informations recueillies ne puissent plus être assimilées par les structures mentales de manière harmonieuse et qu'une rupture solutionne ce déséquilibre, sous la forme de l'accession à un niveau de compréhension supérieur.

6. Ce comportement peut en outre être dû à la fatigue ou à la simple intériorisation de leur rôle d'étudiant.
7. Précision de l'énoncé (listage des informations nécessaires, des résultats attendus), puis raffinement progressif du problème (discernement des objectifs secondaires) et, enfin, choix des structures de programmation pour chaque module.

RÉFÉRENCES

HERMANS, A. (1991) : *Vocabulaire de la sociologie*, Verviers, Marabout.

Actes du deuxième colloque francophone sur la didactique de l'informatique. Facultés Universitaires Notre-Dame de la Paix à Namur, 30, 31 août et 1er septembre, (1990), Presses Universitaires de Namur, AFDI, CeFIS, Namur.

DE SCHAETZEN, C. (1990a) : *Guide complet de WORD 5*, Paris, Eyrolles.

DE SCHAETZEN, C. (1990b) : *WORD 4 et 5. Fonctions avancées : les macro-commandes*, Verviers, Marabout.

Colloque francophone sur la didactique de l'informatique. Université René Descartes, Paris, 1, 2 et 3 septembre 1988, (1989), Paris, Enseignement Public et Informatique.