Le Design Logiciel

Philippe Aigrain

Cet article fut publié à l'origine dans la revue Axe Sud, n° d'Avril-Mai-Juin 1985

Il existe un nombre croissant de biens de consommation qui soit sont des programmes informatiques, soit incorporent des programmes informatiques. La façon dont ces programmes ont été conçus et réalisés, devient un facteur essentiel du succès de ces produits auprès du public, et de l'évaluation que l'on peut faire de leur intérêt culturel. Du point de vue de l'utilisateur d'une machine informatique ou informatisée, il devient de plus en plus difficile de distinguer ce qui relève des programmes qui y sont implantés. D'une façon générale, l'utilisateur ne voit la machine qu'à travers les programmes, elle ne devient directement visible que par ses pannes, ses insuffisances (lenteur) ou les malfaçons de ses programmes. Plus récemment, on a vu apparaître des machines dont certains éléments matériels visibles (souris par exemple) ou invisibles (organisation physique de la mémoire) sont directement fondés sur une recherche concernant l'apparence externe que doit avoir leur utilisation sous contrôle de programmes. On peut dire que le logiciel s'empare du matériel : la campagne de promotion de Macintosh d'Apple est entièrement fondée sur ce qu'on peut appeler son « design logiciel ».

Une prise de conscience de l'importance de ce type de facteurs se développe partout. Cependant, l'abord qui prévaut actuellement en France, est remarquablement étriqué, du fait des caractéristiques de l'industrie et de la recherche française en informatique. Cet article voudrait contribuer à élargir le champ des réflexions dans le domaine.

L'informatique s'est longtemps développée à l'écart du grand public, dans les univers de l'armée, de la gestion des entreprises, des laboratoires, et dans une moindre mesure, de la production industrielle. Il existe de plus une répugnance traditionnelle des gros industriels français à la prise en compte des contraintes spécifiques de la conception de produits grand public sur des marchés non protégés. L'utilisateur grand public est souvent perçu comme un utilisateur professionnel en plus bête et moins familier avec la machine. De ce fait, on renforce simplement lorsqu'on s'adresse à lui, une approche en termes de normes, de qualité et d'efficacité (sécurité, faible coût, ergonomie, etc…) de la production logicielle. C'est cela que l'on appelle le génie logiciel. Bien sûr, il existe de nombreuses équipes ou entreprises que leurs intérêts, ou leur exposition à la concurrence poussent à effectuer des réalisations tout à fait intéressantes du point de vue du design logiciel, mais ces démarches ne trouvent pas de relais dans la recherche ou la formation, elles ne contribuent pratiquement pas à la diffusion d'un savoir-faire. Cette situation conduit à poser deux questions : Pourquoi l'approche normative en matière de logiciels a-t-elle autant de poids ? Sur quelles bases une approche en termes de design pourrait-elle se développer ? Ces questions sont d'autant plus importantes que contrairement à un discours souvent entendu (« les jeunes doivent apprendre l'informatique comme une seconde langue ») on est loin d'avoir atteint une définition claire et unique des moyens et des méthodes souhaitables en matière de logiciels. Au contraire, nous vivons l'ère de l'éclatement, de la multiplication des langages, des types d'interaction avec l'utilisateur.

L'approche normative, séquelle de la préhistoire du logiciel

Pour comprendre le poids de l'approche normative, qui cherche à définir **une** bonne façon de faire du logiciel, un petit parcours historique est nécessaire. En 1951, une dizaine de machines

fonctionnaient dans le monde, à qui on peut plus ou moins attribuer rétrospectivement le nom d'ordinateurs (8 aux Etats-Unis, 2 en Grande-Bretagne). Ces machines étaient en général dédiées à des activités particulières (calcul météorologique, balistique, statistique, scientifique, etc.), même si elles étaient capables d'effectuer d'autres types de calculs. Les moyens de communication de ces machines avec l'extérieur étaient des plus réduits, même pour l'introduction des programmes ; ils consistaient en des lecteurs de rubans perforés, de cartes perforées, des fiches que l'on enfichaient dans des trous aux endroits appropriés, et exceptionnellement (machines de Stiblitz aux Bell Laboratoires), un télétype relié à l'ordinateur. On évitait soigneusement toute forme d'interaction pendant le déroulement d'un programme, qui, dans l'état de la technique d'alors, aurait considérablement ralenti et perturbé le fonctionnement de la machine.

La question d'alors n'était donc pas le dialogue homme-machine, mais seulement la commande de la machine par un programme. John Backus, inventeur du FORTRAN et promoteur actuel de la programmation fonctionnelle, a raconté dans un article à quoi cela ressemblait de programmer dans les années 50. Cela tenait plus de l'exercice de survie dans un milieu hostile et imprévisible, que de l'application de règles de travail éprouvées. La programmation a commencé comme un art, « Une astuce pour chaque problème », était sa devise. Chaque programme était une pièce unique, et l'on repartait souvent de zéro pour faire le suivant. Une majorité des artistes programmeurs se recrutaient chez les femmes, au départ du fait de leur plus grande disponibilité pendant la guerre, mais la tendance s'est maintenue dans l'après-guerre.

Programmer consistait alors à aligner des « instructions » dans ce qu'on appelait à l'époque du « code-machine », suites de 0 et de 1 particulièrement peu expressives du point de vue humain, mais que la machine est capable de comprendre et d'exécuter. Depuis, les machines n'ont d'ailleurs pas évolué de ce point de vue, mais les outils que l'on s'est donné pour leur parler font la différence. Passer d'un problème (« calculer les tables de tir pour tel type d'obus » pour prendre un exemple d'époque) à un programme, était un processus long et rébarbatif. Mais la vraie difficulté commençait quand on essayait de faire « tourner » le programme. Il n'existait aucune aide pour trouver les raisons du non-fonctionnement d'un programme. Il fallait se livrer à ce qu'un spécialiste appela « la lecture dans les entrailles », c'est-à-dire fouiller dans la visualisation de tout le contenu de la mémoire, gigantesque collection de 0 et de 1, juste avant que l'incident ne se produise. En l'absence de possibilité ou de réussite de la lecture dans les entrailles, il restait à modifier le programme jusqu'à ce que quelque bien s'en suive.

Les premiers programmeurs bénéficiaient du délicieux pouvoir qui s'attache à ceux qui possèdent une spécialité rare et mystérieuse. La plupart d'entre eux n'étaient que des utilisateurs formés sur le tas par la dure nécessité. Loin de publier les « trucs » qu'ils inventaient dans la presse technique, ils les gardaient secrets.

Cette situation était bien sûr tout à fait incompatible avec une large diffusion des ordinateurs chez ces clients méfiants que sont les grandes entreprises et les administrations. Un important effort de recherche et de développement fut donc entrepris, principalement sur l'initiative des constructeurs, IBM en tête, pour que la programmation cesse d'être un art, pour devenir une technique, et qui sait une « industrie ».

Les tenants actuels du « génie logiciel » ont une façon à eux de raconter l'histoire des 30 années qui suivent : ils y voient un progrès constant vers la définition d'outils toujours plus sûrs et performants, toujours plus intégrés les uns avec les autres, pour culminer aujourd'hui dans la création des ateliers intégrés de production logicielle. Ainsi on aurait inventé des langages de programmation de mieux en mieux définis, du couple FORTRAN COBOL, en passant par

ALGOL et PASCAL, jusqu'à ADA, le dernier choisi par l'armée américaine. Dans le même temps, les environnements de développement (c'est-à-dire tout ce qu'il y a autour d'un langage pour aider à s'en servir, du système d'opérations de la machine aux éditeurs en passant par les « débuggers ») auraient suivi la même courbe ascendante de progrès, et permettraient de traiter aujourd'hui jusqu'à la spécification des problèmes à traiter par l'informatique et même la mesure de la qualité des logiciels. Dans le même ordre d'idées, la communication homme-machine fait également l'objet d'une approche normative, visant à dégager les principes systématiques de présentation de l'information à l'utilisateur, les moyens par lesquels il convient de solliciter ses actions, etc…

Cette façon de raconter l'histoire et d'organiser le présent du logiciel est inexacte et dangereuse. Elle consisterait, si elle était adoptée partout, à généraliser des principes d'organisation issus de l'univers militaire ou des très gros industriels de l'aéronautique, de l'énergie et de l'espace. Ces organisations réalisent des développements logiciels gigantesques, dans lesquels des centaines de programmeurs différents sont impliqués, qui vont aboutir à des programmes dont la durée de vie est souvent de 15 ou 20 ans, et qui ne sont pas vendus sur un marché d'utilisateurs (ceux qui décident de les acheter ne sont pas ceux qui vont les utiliser). Dans de telles conditions, il n'est pas surprenant qu'une approche normative, cherchant à trouver **le** bon langage, **la** bonne communication homme-machine, se soit imposée. Mais si l'on considère l'ensemble des domaines d'application de l'informatique, le paysage devient tout autre.

Le cas des langages de programmation est parlant : il en existe des dizaines aujourd'hui, et un des gros problèmes de la communauté scientifique consiste à empêcher chaque chercheur qui a une idée, d'inventer immédiatement un langage de programmation pour la mettre en œuvre. Chacun de ces langages se révèle être bon … à quelque chose, c'est-à-dire à un certain type d'utilisateur et d'utilisation. Cela ne signifie nullement qu'il est impossible de faire des choix, de critiquer tel ou tel langage, mais que les critères à mettre en œuvre sont multiples et contradictoires. Parmi les langages les plus utilisés, certains comme Basic, APL ou LISP sont absents de la fameuse trajectoire du progrès supposé, mais ont leurs adeptes farouches.

Mais le cas de la communication homme-machine est plus important encore, car il concerne l'ensemble des utilisateurs de l'informatique ou des objets techniques informatisés, il touche l'«apparence » externe des logiciels et non le seul programmeur. Là encore, il ne saurait être question de négliger l'accumulation d'outils et de savoir-faire des 30 dernières années, mais de mesurer qu'il s'agit d'un développement polymorphe et que les choix à effectuer dans la conception d'un produit qui se veut aujourd'hui culturel ne sont pas de l'ordre de l'application de normes de qualité ou d'efficacité, mais bien de choix culturels. Pour donner un exemple du type de choix à effectuer dans la création d'un logiciel, choix dont la composition constitue le design logiciel, la suite de l'article explore une série d'oppositions qui partagent différents types de logiciels, et plus généralement d'objets techniques.

La Boîte Noire et la Maison de Verre

Un premier type d'oppositions prend forme autour de la question : "un programme doit-il rendre visible le processus et les étapes intermédiaires par lesquels il passe de ses données à ses résultats ?". Dans le modèle de la « boîte noire », l'utilisateur ne doit rien voir entre ses actions et leurs résultats recherchés : on parle d'utilisation transparente de l'objet technique, parce que l'on ne voit pas l'objet technique lui-même, mais seulement ce qu'il y a « derrière », le but pour lequel on l'utilise. Dans le modèle de la « maison de verre », le programme rend au contraire visible le processus qu'il effectue pour passer de données à des résultats. Cette visualisation

se fait en général en montrant des étapes intermédiaires, ou bien en visualisant un certain nombre de variables d'état. Le modèle de la boîte noire correspond en général à une recherche d'économie des actions de l'utilisateur, alors que celui de la maison de verre correspond à une recherche de contrôle fin, ou à un mécanisme d'élaboration progressive d'une stratégie. Un exemple non-informatique est celui de l'automobile : entre la boîte de vitesses automatique, et la boîte de vitesses classique agrémentée d'un compte-tours, on peut choisir différents degrés de mise en évidence pour le conducteur du processus mécanique.

Les spécialistes de la diffusion de l'informatique ont souvent tendance à considérer que seul le modèle de la « boîte noire » est compatible avec des utilisations grand public. Ils basent cette opinion sur le fait que la plupart des objets techniques qui se sont largement diffusés (téléphone, télévision, par exemple) sont du type Boîte Noire. Avoir à se confronter au processus technique, fut-ce à travers une visualisation tout à fait détournée, est vu comme une nuisance acceptable par les seuls spécialistes. Outre que l'exemple déjà cité de l'automobile devrait relativiser cette affirmation, le cas des programmes informatiques est tout à fait spécial. Un programme informatique sert à manipuler des symboles, et à manipuler des objets par le biais de cette manipulation des symboles. De ce fait, il est possible que les états intermédiaires d'un programme soient « de même nature » que ses résultats. Ainsi, il est possible de s'y confronter sans avoir à entrer dans les détails de la quincaillerie technique. De plus, il est fréquent que ce que l'on cherche, et le chemin à parcourir pour y parvenir, ne se définissent que progressivement au cours de l'utilisation. La personne qui interroge une banque de données pour y découvrir la maison de vacances où passer une semaine est peu susceptible de dire à priori que celle-ci répond à tel ou tel critère. Il est bien plus probable que confrontée à l' « image » d'une maison (réelle ou fictive), elle puisse énoncer ce qui lui y plaît et lui y déplaît et se promener ainsi de maison en maison, jusqu'à « arrêter » son choix.

La Maison de Verre représente donc un modèle aussi valide que la boîte noire pour certains programmes. Mais qui plus est, le choix entre les deux modèles n'est pas d'ordre fonctionnel. Il ne suffit pas d'examiner le « problème » et de déterminer si ses caractéristiques relèvent de l'une ou l'autre approche, une certaine part de choix « stylistique » est nécessaire. C'est ce qui justifie une approche en termes de design, et non un simple approfondissement de l'approche normative.

Question de rythme

Pendant longtemps, la question du rythme d'un programme interactif s'est posée dans des termes très simples : ces sacrées machines répondaient toujours trop lentement ! On a développé un certain nombre de règles en matière de réactions d'un programme aux actions de l'utilisateur : notifier immédiatement que l'action de l'utilisateur a bien été enregistrée, rendre ses effets les plus rapides possibles, et si le traitement ne peut être bref, signaler d'une façon ou d'une autre qu'il est en train de se dérouler de façon à ce que l'utilisateur sache qu'on s'occupe de lui … Mais la question du rythme se pose aussi d'une toute autre manière : à quel rythme demande-t-on à l'utilisateur d'agir ? tous les combien de temps sollicite-t-on ses actions ? Combien de temps est-ce qu'on lui laisse pour les effectuer ?

Les jeux vidéos donnent un exemple de rythme d'interaction où l'on recherche le tempo le plus élevé possible, et où ce tempo est entièrement fixé par le programme. Le programme effectue une boucle serrée « saisie des actions de l'utilisateur sur les boutons et leviers de commande / répercussion sur l'état de l'univers du jeu », et si l'utilisateur ne fait rien, cela équivaut à des actions par défaut, qui le plus souvent le conduisent à une perte certaine. Ce type de rythme a l'avantage de capter entièrement l'attention de l'utilisateur, mais cet avantage peut se révéler

être un sérieux inconvénient dans d'autres contextes. Il est parfois utile et agréable de disposer d'autant de temps que l'on le veut pour prendre une décision d'action, ou de pouvoir interagir avec un programme en faisant autre chose en même temps, ou de pouvoir interagir avec plusieurs programmes simultanément. Le tempo d'interaction est un déterminant essentiel du type de pensée qui sera favorisé par un programme. L'exemple de la différence entre les parties d'échecs dites de Blitz (éclair) et les parties normales en est une illustration. Enfin, laisser un temps de réflexion assez long à l'utilisateur suppose qu'on lui présente une information riche, faute de quoi la lassitude est inévitable.

Simulation et reconstruction

Un très grand nombre de programmes informatiques remplissent des fonctions qui ont été ou sont encore remplies par des personnes, ou au moyen d'objets techniques non-informatisés. Le fait que l'informatique manipule des symboles, permet d'envisager des rapports de type très différents entre le programme et les référents extérieurs que sont par exemple, le dialogue parlé et écrit avec un employé de banque pour le programme d'un distributeur automatique de billets, ou l'écriture au moyen d'un style sur une feuille de papier pour un programme de traitement de textes (dans ce cas, l'usage d'une machine à écrire constitue bien sûr un autre référent possible). Le programme peut entretenir deux types de rapports au moins avec ces référents extérieurs : il peut chercher à les « simuler », c'est-à-dire à les reproduire plus ou moins sur un support différent. Cette simulation peut s'exercer à plusieurs niveaux : on peut simplement installer un certain parallélisme d'opérations, ou aller jusqu'à copier dans le détail, le type de gestes qui était effectué dans la situation extérieure, comme cela se fait dans certaines palettes graphiques qui essayent de simuler la gestuelle du peintre. Mais on peut aussi chercher à reconstruire complètement l'activité que réalise le programme informatique, proposer d'arriver à des buts plus ou moins semblables par des moyens complètement distincts. Par exemple, certains programmes de composition musicale prennent avantage de la possibilité de définir dans l'informatique des traitements systématiques et complexes de symboles pour introduire des techniques de composition nouvelle (d'autres simulent l'écriture musicale usuelle). En robotique aussi, les deux modèles sont présents, avec les robots peintres qui copient les gestes d'un manipulateur côté simulation, et de nombreux robots à commande numérique côté reconstruction.

Dans un premier temps, la plupart des programmes informatiques ont fonctionné sous la forme « reconstruction »… parce que l'on ne pouvait pas faire autrement, les moyens techniques ne permettant pas la simulation réussie de situations souvent complexes à représenter. Dans un second temps, des programmes sont apparus, comme le système STAR de Xerox, sur lequel a été copié le système d'opérations du Macintosh, qui sont entièrement fondés sur la simulation. Dans le cas de STAR et Macintosh, on simule sur la machine le fonctionnement d'un bureau physique, avec ses dossiers, sa poubelle, son horloge, etc… L'approche « simulation » a conduit à des succès certains, et facilite l'apprentissage en plaçant l'utilisateur dans une situation « reconnaissable ». On aurait cependant tort de croire qu'elle est une panacée : la liberté, le caractère exploratoire de l'approche « reconstruction » a également des avantages.

Langage naturel et langage formel

Les deux oppositions abordées maintenant ont un rapport étroit avec la précédente, sans pour autant se confondre avec elle. Un important effort de recherche est actuellement entrepris pour la compréhension du langage naturel par les machines. Cet effort bute sur de grosses difficultés, et pose des problèmes philosophiques et épistémologiques de taille. Certains

chercheurs limitent leurs efforts à la compréhension du langage déjà codé sous formes de symboles (entrée au clavier de phrases), d'autres cherchent à reconnaître directement le langage parlé ou écrit. Comme ces chercheurs contribuent de façon passionnante à une meilleure connaissance du langage humain, et que leur enjeu paraît si essentiel, une sorte de doxa s'est créée et plus personne ne semble douter que cela serait un avantage de pouvoir dialoguer avec les machines en langage naturel. Il y a là un cas particulier de la vogue de la simulation, c'est-à-dire que l'on valorise le fait que le rapport homme ou femme/machine soit une simulation du rapport entre deux êtres humains. La plupart des programmes de bases de données vantent aujourd'hui en mentant plus ou moins leurs interfaces « proches du langage naturel ». Or, il n'est pas du tout certain que lorsqu'on doit donner des ordres d'action à une machine informatisée, ce soit systématiquement un avantage de le faire en langage naturel, plutôt que dans un langage formel. On verra plus loin que dans certains cas, on peut très bien se passer complètement de langage de commandes, mais supposons ici qu'il en faille un.

Un langage de commandes sert à indiquer à une machine ce qu'elle doit faire. Il s'agit souvent d'une succession assez complexe d'actions très simples. La qualité d'un langage de commandes est alors un compromis entre la facilité de l'utilisateur à s'exprimer dans ce langage, la concision des ordres quand on les énonce dans ce langage (faire un long discours à une machine peut être très ennuyeux), et la non-ambiguité des ordres énoncés, etc… On peut bien sûr imaginer une solution mi-chèvre – mi-chou, consistant à permettre l'entrée des commandes en langage naturel (en supposant résolu convenablement le problème de sa compréhension, ce qui est plus que douteux pour les prochaines années) ou en langage formel, suivant les préférences ou le savoir-faire de l'utilisateur. On peut aussi aborder ce choix comme un choix « stylistique ».

Ressemblance et codage

L'opposition entre ressemblance et codage concerne les programmes qui sont amenés à présenter à l'utilisateur des représentations d'objets appartenant à un univers de référence, en général parce qu'ils se situent dans l'approche « simulation ». La représentation de ces objets peut être plus ou moins « ressemblante ». Par exemple, pour représenter un dossier contenant des informations, on pourra utiliser une photographie de dossier (transférée sur un écran vidéo) ou, au contraire, un pictogramme très simple, une représentation codée symbolisant un dossier. La ressemblance a l'avantage d'induire une illusion de réalité plus forte, et le codage de permettre des manipulations plus aisées. On trouve un bon éventail de choix dans le domaine des jeux vidéos, entre un jeu comme Pacman, et les jeux dits à image réelle basés sur des vidéodisques. Encore une fois, on aurait tort de considérer qu'il y a progrès évident dans le passage à l'image « réelle », simplement parce que celle-ci s'est trouvée être possible technologiquement quelques années plus tard. Le système du Macintosh ne gagnerait évidemment rien à utiliser des représentations ressemblantes à la place de ses « icones ». Dans les systèmes de parcours d'ensembles d'images basés sur la métaphore du voyage, l'utilisation, au moment des choix proposés à l'utilisateur, d'images du même type que celles parcourues, conduit au choix inverse.

Commande gestuelle et langage de commande

Un nombre important de programmes informatiques sollicitent les actions de leurs utilisateurs non sous la forme de commandes exprimées dans un langage, mais sous la forme de gestes. La commande gestuelle a l'avantage de l'immédiateté, et de la continuité avec les habitudes de commande des objets techniques non-informatisés. La commande par un langage permet la

formulation de commandes complexes, et réutilisables. On évitera le faux débat qui résulte du fait que pour exprimer des ordres à une machine dans un langage, on utilise en général des gestes (par exemple, frapper les touches d'un clavier) et qu'à l'inverse on peut parler de langage gestuel.

Dans de nombreux cas, on peut choisir entre langage de commande et commande gestuelle en analysant ce sur quoi portent les actions de la machine. On privilégiera le langage de commande quand il s'agit d'effectuer des manipulations complexes de signes élémentaires, et la commande gestuelle quand il s'agit d'effectuer des manipulations simples d'objets complexes. Mais cette détermination normative se révélera souvent insuffisante dans des cas « intermédiaires ». Il ne fait pas de doute que ce choix est aussi affaire de goût et d'esthétique : « être aux commandes » et « donner des ordres » sont deux situations dont les connotations sont fort différentes.

Prévisibilité et surprise

Jusqu'à ces derniers temps, lorsqu'une machine nous faisait une surprise, c'était une mauvaise surprise. On a ainsi pris l'habitude de considérer qu'il était toujours souhaitable de pouvoir prévoir quelle serait la réaction d'un programme à une action de son utilisateur. Les premiers contre-exemples à cette règle sont venus du domaine des jeux : dans un jeu comme Adventure, par exemple, une grande partie du plaisir de jouer vient de l'imprévisibilité de réactions du programme aux ordres qui lui sont donnés. On pouvait encore croire, vu la spécificité du domaine des jeux, à une exception confirmant la règle. Puis on a vu surgir des réflexions sur la notion de programme indéterministe dans l'informatique théorique, ou des applications audiovisuelles interactives qui réservent bien des surprises (pas forcément mauvaises) à leurs utilisateurs. D'une façon plus générale, dans toutes les applications où l'utilisateur n'est pas sensé savoir ce qu'il veut, ou s'il le sait, n'est pas sensé avoir trop d'idées sur les moyens d'y parvenir, la surprise peut être un ressort utile et agréable.

Le design graphique informatique

La conception graphique des écrans ou des sorties imprimées des programmes informatiques met bien sûr en œuvre des techniques voisines de celles du design graphique en général. Mais une petite histoire vraie, contée ci-après fera percevoir à quel point la perception globale qu'apporte le design logiciel est nécessaire dans ce domaine également.

Lors de l'apparition du Minitel, les protestations fusèrent de toutes parts contre les limitations techniques de cet appareil qui ne permettait que l'utilisation des graphismes rudimentaires du fait de la technique alpha-mosaïque qui y est employée. En réaction à ces critiques, l'Office d'Annonces (une filiale du groupe Havas) et également quelques sociétés de services, embauchèrent pour le vidéotexte (dont le Minitel est l'une des formes) des graphistes professionnels. Un travail important fut entrepris, et effectivement, aboutit à la réalisation d'écrans graphiques qui tiraient part des limitations même de l'appareil pour obtenir des effets originaux et, ma foi, réussis. Ces écrans furent abondamment photographiés, reproduits dans la presse, présentés à des chefs d'Etats et loués à tout point de vue. Puis, on essaya d'introduire dans des applications télématiques ces écrans au graphisme élaboré. Lors de leur première utilisation, les usagers de l'application télématique étaient ravis de l'esthétique des écrans, mais lors d'utilisations régulières, ils finissaient par les prendre en horreur. En effet, plus le graphisme d'un écran est élaboré, plus son dessin sur l'écran est lent du fait de la lenteur de la transmission téléphonique. Or, la structure des applications télématiques imposait

le passage répété par les mêmes écrans.

A l'heure actuelle, les peintres et les graphistes en question utilisent le savoir-faire acquis lors de cette expérience en travaillant sur des outils beaucoup plus performants (type palette Graph-8) et fabriquent des images à destination des supports imprimés ou vidéo. Quant aux applications télématiques, elles se contentent de quelques « logos » sophistiqués, et d'un graphisme minimal.

Plus généralement, la conception graphique n'est pas un domaine indépendant du reste du design logiciel. Elle n'est pas le petit domaine réservé où les points de vue artistique ou culturel auraient droit de cité, tout le reste de la conception des programmes relevant d'une approche purement technicienne.

Vers le design logiciel

Il y a donc une place pour le design logiciel, il y a des questions qui se posent à lui, des choix à faire, des styles à créer, des gens à séduire. Mais au fait, où rencontre-t-on cette bête ? Et bien, on ne la rencontre pas. Le design logiciel n'est qu'une idée, mais cela n'interdit pas de dessiner les contours qu'il pourrait prendre si, par aventure, on décidait de faire de cette idée un projet.

La pire des façons d'y parvenir serait de croire qu'il s'agit d'un métier, et à la façon bien française, de créer une école pour y former de futurs designers logiciels, avant même que l'on sache quels sont les savoirs et les techniques effectivement mobilisés dans cette activité. Une phase d'expérimentation est inévitable, dans laquelle l'essentiel est de mettre en contact des gens de spécialités diverses, et d'intérêts suffisamment éclectiques, avec la réalité de projets de création de programmes informatiques. Il ne faut pas se cacher que la difficulté essentielle consiste à trouver des créateurs qui voient dans ce domaine un véritable enjeu de création. Ce ne pourra être le cas que si une liberté suffisante est laissée, dans un contexte expérimental, pour la définition d'objets, de concepts informatiques originaux. N'oublions pas que le système STAR de Rank Xerox, dont le Macintosh n'est qu'un sous-produit, est le résultat d'années de travail d'une équipe pluridisciplinaire, et que ce système fut un relatif échec commercial, qui a surtout profité … à ses concurrents. Dans ce domaine, comme dans tant d'autres qui mêlent technique et culture, il faut avoir le courage de ne pas se faire trop d'illusions sur les retombées économiques à court terme de cet effort de recherche.

Il y a cependant des bénéfices plus immédiats que la collectivité peut attendre de l'expérimentation dans ce domaine. Le principal est celui de la formation d'un esprit critique, d'un espace de discussion où ce genre de problématique ait droit de cité. L'essentiel des projets importants en matière d'informatisation est mis en œuvre en France par des administrations publiques ou financé par elles. Ces administrations, loin d'exercer un pouvoir omniprésent et tentaculaire, sont soumises à un réseau de pressions économiques et institutionnelles qui ont le don d'orienter leurs projets vers des choix plus que critiquables du point de vue culturel. La rubrique scientifique d'Axe Sud contribuera prochainement à la formation de cet esprit critique à propos des réseaux câblés et de l'expérience de Biarritz, de l'informatisation des écoles et d'autres sujets.

Enfin, il faut lever une ambiguïté que le ton polémique de cet article à propos du génie logiciel a pu faire surgir. Le design logiciel ne se développera certainement pas en ignorant le savoir-faire technique de l'informatique moderne, pas plus qu'un designer de mobilier ne peut ignorer les techniques de production contemporaines. Il s'agit au contraire d'injecter dans l'univers

technique des exigences nouvelles.