L'informatique comme un art

Donald E. Knuth

Conférence de remise de récompense

[La citation pour la remise de la récompense Turing a été lue par Bernard A. Galler, directeur du Comité de remise de la récompense Turing en 1974, en présentation de cette conférence le 11 novembre à la conférence annuelle de l'ACM à San Diego.]

La récompense A.M. Turing de l'ACM est attribuée chaque année à un individu sélectionné par l'ACM pour ses contributions de nature technique faites à la communauté informatique. En particulier, ces contributions devraient avoir une influence significative sur un segment majeur du domaine de l'informatique.

"La récompense A.M. Turing pour l'année 1974 est attribuée au Professeur Donald E. Knuth de l'Université de Stanford pour ces nombreuses contributions à l'analyse des algorithmes et à la conception des langages de programmation, et en particulier à ses notables contributions à l'"Art de la programmation des ordinateurs" à travers sa série de livres très renommés. Les collections de techniques, d'algorithmes et de théorie dans ces livres a servi de point focal pour développer les curricula informatiques et a influencé l'organisation de l'informatique."

Une telle présentation formelle ne met pas réellement en perspective le rôle que Don Knuth a joué en informatique, ainsi que dans l'industrie des ordinateurs vue comme un tout. Cela a été mon sentiment par rapport au premier récipiendaire de la récompense Turing, le Professeur Alan J. Perlis, qui à chaque meeting auquel il participe réussit à fournir l'éclairage sur les problèmes discutés qui devient le point focal de la discussion pour le reste de la rencontre. D'une façon très similaire, le vocabulaire, les exemples, les algorithmes, et l'éclairage que Don Knuth a fournis dans son excellente collection de livres et articles a commencé à trouver son chemin dans un très grand nombre de discussions dans la plupart des champs du domaine. Cela n'advient pas aisément. Comme tout auteur le sait, même un seul livre nécessite une grande quantité d'une organisation sans faille et un dur travail. Tous peuvent apprécier la vision claire et la patience et l'énergie avec lesquels Knuth a pu planifier sept volumes et comment il a pu mener à bien son plan si précautionneusement et de façon si approfondie.

Il est remarquable que cette récompense et les autres qu'il a déjà reçues lui soient données après que trois volumes de cette œuvre aient été publiés. Nous sommes maintenant prêts à dire à tous combien nous apprécions le dévouement de Don Knuth et ses contributions à notre discipline. Je suis très heureux d'avoir présidé le Comité qui a choisi Don Knuth comme le récipiendaire de la récompense A.M. Turing Award de l'ACM en cette année 1974.

Quand les Communications de l'ACM ont commencé à publier en 1959, les membres du comité éditorial de l'ACM firent la remarque suivante alors qu'ils décrivaient les objectifs des périodiques de l'ACM [2] : "Si la programmation des ordinateurs doit devenir une partie importante de la recherche et du développement des ordinateurs, une transition faisant passer la programmation du statut d'art à celui de discipline scientifique doit avoir lieu." Un tel objectif a été un thème toujours récurrent les années suivantes ; par exemple, nous lisons en 1970 les "premières étapes en vue de transformer l'art de la programmation en une science" [26]. Pendant tout ce temps, nous avons

Communications de l'ACM, Décembre 1974, Volume 17, numéro 12.
Adresse de l'auteur, Département d'informatique, Université de Stanford, Stanford, CA 94305

vraiment réussi à transformer notre discipline en science, et d'une façon remarquablement simple : seulement en décidant de l'appeler la "science des ordinateurs".

Sous-jacente à ces remarques, il y a l'idée qu'il n'est pas très désirable qu'un domaine de l'activité humaine soit classé comme un "art" ; il doit être une science avant d'acquérir tout statut réel. D'un autre côté, j'ai travaillé plus de 12 ans sur une série de livres appelés "L'*Art* de la programmation des ordinateurs." Les gens me demandent souvent pourquoi j'ai choisi un tel titre ; et en fait, quelques personnes apparemment ne croient pas que je l'aie vraiment fait, puisque j'ai vu au moins une référence bibliographique à mes livres sous la forme l'"Acte de la programmation des ordinateurs."

Dans cet exposé, j'essaierai d'expliquer pourquoi je pense que le mot "Art" est le mot approprié. Je discuterai de ce que cela signifie pour quelque chose d'être un art, par opposition au fait d'être une science ; j'essaierai d'examiner si les arts sont de bonnes ou de mauvaises choses ; et j'essaierai de montrer qu'un point de vue approprié sur ce sujet nous aidera à améliorer la qualité de ce que nous faisons.

L'une des premières fois où j'ai été interrogé à propos du titre de mes livres eut lieu en 1966, durant le meeting national précédent de l'ACM qui eut lieu dans le sud de la Californie. C'était avant qu'aucun de ces livres n'ait été publié et je me rappelle avoir déjeuné avec un ami à l'hôtel du colloque. Il savait combien j'étais vaniteux, déjà à cette époque, alors il me demanda si j'allais appeler mes livres "Une introduction à Don Knuth." Je lui répondis que, au contraire, j'avais appelé ces livres comme une introduction à *lui*. Son nom était Art Evans, (L'Art de la programmation des ordinateurs en personne.)

De cette histoire, nous pouvons conclure que le mot "art" a plus d'un sens. En fait, un des plus belles choses au monde est qu'il est utilisé selon de nombreux sens différents, chacune de ces acceptions étant assez appropriée pour la programmation des ordinateurs. Lorsque j'ai préparé cet exposé, je suis allé à la bibliothèque pour trouver ce que les gens avait écrit à propos du mot "art" au fil des ans ; et après avoir passé quelques jours fascinants parmi les piles de livres, j'en vins à la conclusion que le mot "art" doit être l'un des mots les plus intéressants de la langue anglaise.

Les arts des anciens

Si l'on revient aux racines latines, on trouve qu'*ars, artis* signifie "compétence." Cela est sûrement significatif que le mot grec correspondant soit $\tau\acute{\epsilon}\chi\nu\eta$, la racine à la fois du mot "technologie" et du mot "technique."

De nos jours, quand on parle d'"art", on pense probablement d'abord aux Beaux-Arts, comme la peinture et la sculpture, mais avant le vingtième siècle, le mot était généralement utilisé dans un sens plutôt différent. Puisque cet ancien sens de "art" perdure dans de nombreux idiomes, particulièrement lorsque nous opposons l'art et la science, je voudrais prendre quelques minutes pour vous parler de l'art au sens classique du terme.

Au Moyen-Âge, les premières universités furent créées pour enseigner les "arts libéraux", que sont

la grammaire, la réthorique, la logique, l'arithmétique, la géométrie, la musique et l'astronomie. Notez combien cela diffère du curriculum enseigné dans les lycées d'arts libéraux de nos jours, ainsi que la manière dont trois des sept arts libéraux initiaux sont des composants importants de l'informatique. À cette époque, un "art" était une activité conçue par l'intellect humain, par opposition aux activités dérivées de la nature ou de l'instinct ; les arts "libéraux" étaient libérés, ou libres, par opposition aux arts manuels, comme le labour (cf. [6]). Au Moyen-Âge, le mot "art" lui-même désignait habituellement la logique [4], qui était alors l'étude des syllogismes.

Science vs. Art

Le mot "science" semble avoir été utilisé pendant de nombreuses années à peu près dans le même sens que le mot "art" ; par exemple, les gens parlaient également des sept sciences libérales, qui étaient identiques aux sept arts libéraux [1], Duns Scotus au treizième siècle appelait la logique "la Science des Sciences, et l'Art des Arts" (cf. [12, p. 34f]). Alors que la civilisation et les connaissances se développaient, les mots prirent des significations de plus en plus indépendantes, le mot "science" étant utilisé pour désigner les connaissances, et le mot "art" pour l'application des connaissances. Ainsi, la science astronomique était la base de l'art de la navigation. La situation ressemblait à la manière dont aujourd'hui, nous faisons la distinction entre la "science" et l'"ingénierie."

De nombreux auteurs ont écrit à propos des relations entre art et science au dix-neuvième siècle, et je crois que les meilleures arguments ont été donnés par John Stuart Mill. Il a dit les choses suivantes, parmi d'autres, en 1843 [28] :

> Plusieurs sciences sont souvent nécessaires pour former le socle d'un seul art. Telle est la complexité des affaires des hommes, qui permet que pour qu'une seule chose soit *faite*, il est souvent nécessaire de *connaître* la nature et les propriétés de nombreuses choses... L'art en général consiste à assembler les vérités de la science, de la manière la plus efficace pour que cet art soit pratiqué, plutôt que de la manière la plus efficace pour que cet art soit pensé. Les domaines scientifiques groupent et arrangent leurs vérités de manière à nous permettre d'appréhender autant que possible d'un seul regard l'ordre général de l'univers. L'art... assemble les vérités de différents domaines de la science éloignés les uns des autres, vérités liées à la production de conditions différentes et hétérogènes nécessaires à chaque effet que nécessitent les exigences de la vie pratique.

Comme je regardais ces choses à propos de la signification du mot "art," je trouvai que les auteurs parlaient d'une transition de l'art à la science durant au moins deux siècles. Par exemple, on trouve dans la préface d'un livre de minéralogie, écrit en 1784, la phrase suivante [17] : "Avant l'année 1780, la minéralogie, bien qu'elle ait pu vraiment être perçue par beaucoup comme un art, pouvait difficilement être considérée comme une science."

Selon la plupart des dictionnaires, le mot "science" signifie une connaissance qui a été logiquement organisée et systématisée sous la forme de "lois" générales. L'avantage de la science est qu'elle nous préserve de la nécessité de penser aux choses dans chaque cas particulier ; nous pouvons orienter nos pensées vers des concepts de plus haut niveau. Comme l'a écrit John Ruskin en 1853 [32] : "Le travail de la science consiste à remplacer les apparences par des faits, et les impressions par des

démonstrations."

Il me semble que si les auteurs que j'ai étudiés écrivaient de nos jours, ils seraient d'accord avec la caractérisation suivante : la science est une connaissance que nous comprenons si bien que nous pouvons l'inculquer à un ordinateur ; et si nous ne comprenons pas correctement quelque chose, c'est un art que d'y faire face. Puisque la notion d'algorithme et de programme d'ordinateur nous fournit un test extrêmement utile pour connaître la profondeur de notre connaissance de n'importe quel sujet, le processus consistant à aller de l'art vers la science signifie que nous apprenons à automatiser ce processus.

L'intelligence artificielle a fait des progrès significatifs, même s'il reste un écart énorme entre ce que les ordinateurs pourront faire dans un futur proche et ce que les gens ordinaires sont capables de faire. Les processus mystérieux mis en œuvre par les personnes lorsqu'elles parlent, écoutent, créent, et même lorsqu'elles programment, sont hors de portée de la science ; presque tout ce que nous faisons est encore artisanal.

De ce point de vue, il est certainement souhaitable de faire de l'informatique une science, et nous avons en effet suivi un long chemin pendant ces 15 années nous séparant de la publication des remarques que j'ai citées au début de mon intervention. Il y a quinze ans, la programmation des ordinateurs était si mal comprise que presque personne ne pensait à prouver que les programmes étaient corrects ; nous nous dépatouillons avec un programme jusqu'à ce que nous "sachions" qu'il marchait. À ce moment-là, nous ne savions même pas comment exprimer le *concept* de correction d'un programme, d'une façon rigoureuse quelle qu'elle soit. C'est seulement dans les années récentes que nous avons appris le processus d'abstraction par lequel les programmes sont écrits et compris ; et cette nouvelle connaissance à propos de la programmation est en train de produire de gros gains en pratique, même si peu de programmes sont vraiment prouvés comme étant corrects selon une parfaite rigueur, puisque nous commençons à comprendre les principes de la structure des programmes. Le point important est que quand nous écrivons des programmes aujourd'hui, nous savons que nous pourrions en principe construire des preuves formelles de leur correction si nous le voulions vraiment, maintenant que nous comprenons comment de telles preuves doivent être écrites. Cette base scientifique résultent dans des programmes plus robustes que ceux que nous écrivions aux premiers jours de l'informatique quand l'intuition était la seule base étayant la correction des programmes.

Le domaine de la "programmation automatique" est l'un des champs de recherche les plus vastes de l'intelligence artificielle aujourd'hui. Ses partisans rêveraient de donner une conférence intitulée "la programmation des ordinateurs comme artefact" (signifiant que la programmation serait simplement devenue une relique des jours passés), parce que leur objectif est de créer des machines qui écriraient des programmes mieux que nous, en leur donnant seulement les spécifications d'un problème. Personnellement, je ne pense pas qu'un tel but soit jamais atteint, mais je pense que leur recherche est extrêmement importante, parce que tout ce que nous apprenons au sujet de la programmation nous aide à améliorer notre propre art. En ce sens-là, nous devrions sans cesse nous efforcer de transformer *tout* art en une science : dans ce processus, nous faisons avancer l'art.

Science et Art

Notre exposé montre que la programmation des ordinateurs est maintenant *à la fois* une science et un art, et que les deux aspects se complémentent joliment l'un l'autre. Apparemment la plupart des auteurs qui étudient un telle question aboutissent à cette même conclusion, que leur sujet est à la fois une science et un art, quel que soit leur sujet (cf. [25]). J'ai trouvé un livre au sujet de la photographie, écrit en 1893, qui énonçait que "le développement de l'image photographique est à la fois un art et une science" [13]. En fait, quand j'ai ouvert un dictionnaire pour la première fois pour étudier les définitions des mots "art" et "science," j'ai jeté un œil aux préfaces des éditeurs, qui commençaient par dire "L'écriture d'un dictionnaire est à la fois une science et un art." L'éditeur du dictionnaire Funk & Wagnall [27] a observé que l'accumulation méticuleuse et la classification des données à propos des mots a un caractère scientifique, dans la mesure où l'écriture de définitions contenant des mots bien choisis nécessite d'être capable d'écrire avec économie et précision : "La science sans l'art est susceptible d'être inefficace ; l'art sans la science est de façon certaine inapproprié."

En préparant cette conférence, j'ai parcouru le catalogue des cartes à la bibliothèque de Stanford pour voir comment d'autres personnes avaient utilisé les mots "art" et "science" dans les titres de leurs livres. Cela s'est avéré assez intéressant.

Par exemple, j'ai trouvé deux livres intitulés *L'art de jouer du piano* [5, 15], et d'autres appelés *La science de la technique du pianoforte* [10], *La science de la pratique du pianoforte* [30]. Il y a également un livre appelé *L'art de jouer du piano : une approche scientifique* [22].

J'ai aussi trouvé un joli petit livre intitulé *L'art doux des mathématiques* [31], qui m'a rendu assez triste parce que je ne peux honnêtement décrire la programmation des ordinateurs comme un "art doux".

Je connaissais depuis plusieurs années un livre dont le titre est *L'art du calcul*, publié à San Francisco, en 1879, par un homme nommé C. Frusher Howard [14]. C'était un livre sur l'arithmétique pratique pour les affaires qui avait été vendu à 400 000 exemplaires dans plusieurs éditions depuis 1890. J'avais été amusé par la lecture de la préface de ce livre, puisqu'elle montre que la philosophie d'Howard et l'intention de son titre étaient quelque peu différentes des miennes ; il écrit : "La connaissance de la science des nombres est d'une importance mineure ; la compétence dans l'art du calcul est absolument indispensable."

Plusieurs livres mentionnent à la fois la science et l'art dans leur titre, notamment *La science d'être et l'art de vivre* de Maharishi Mahesh Yogi [24]. Il y a aussi un livre dont le titre est *L'art de la découverte scientifique* [11], qui analyse comment ont été faites quelques unes des grandes découvertes de la science.

Voici tout ce qu'on peut trouver à propos du mot "art" dans son acception classique. Vraiment lorsque j'ai choisi ce titre pour mes livres, je ne pensais pas en premier lieu à l'art dans ce sens-là, je pensais davantage à ses acceptions courantes. Le livre probablement le plus intéressant qui se présenta à moi durant ces recherches est un livre relativement récent de Robert E. Mueller intitulé

La science de l'art [29]. De tous les livres que j'ai mentionnés, celui de Mueller est le plus proche de ce que je souhaitais être le thème central de mon allocution aujourd'hui, par rapport à l'art tel que nous entendons ce terme maintenant. Il observe : "On a un jour pensé que les perspectives imaginatives de l'artiste signaient la mort du scientifique. Et que la logique de la science semblait jeter un sort à tous les vols artistiques de la fantaisie." Il continue à explorer les avantages qui résultent effectivement d'une synthèse entre la science et l'art.

Une approche scientifique est en général caractérisée par les mots logique, systématique, impersonnelle, calme, rationnelle, alors qu'une approche artistique est caractérisée par les mots esthétique, créative, humanitaire, anxieuse, irrationnelle. Il me semble que ces deux approches apparemment contradictoires sont très appropriées pour qualifier la programmation des ordinateurs.

Emma Lehmer a écrit en 1956 qu'elle avait trouvé que la programmation était "une science exigeante autant qu'un art intrigant" [23]. H.S.M. Coxeter remarqua en 1957 qu'il se percevait lui-même "plus comme un artiste que comme un scientifique." [7]. C'est à cette époque que C.P. Snow tirait la sonnette d'alarme au sujet de la polarisation grandissante entre "deux cultures" des personnes cultivées [34, 35]. Il insista sur le fait que nous avions besoin de combiner les valeurs scientifiques et artistiques si nous voulions réellement progresser.

Les œuvres d'art

Lorsque je suis auditeur d'une longue conférence, mon attention tend à diminuer à peu près à ce moment dans l'heure. Par conséquent, je me demande si vous commencez à être las de ma harangue à propos de la "science" et de l'"art" ? J'espère vraiment que vous pourrez écouter attentivement ce qui va suivre, parce que c'est maintenant qu'arrive la partie que je ressens la plus profondément.

Quand je parle de programmation des ordinateurs comme un art, je pense d'abord à elle comme à une *forme* d'art, au sens esthétique du terme. Le but principal de mon travail en tant qu'enseignant et auteur est d'aider les gens à apprendre à écrire de *beaux programmes*. C'est pour cette raison que j'étais particulièrement heureux d'apprendre récemment [32] que mes livres étaient présentés dans la bibliothèque des Beaux-Arts à l'Université Cornell. (Pourtant, il semblerait que les trois volumes soient soigneusement restés sur l'étagère, sans être utilisés, et je crains que les bibliothécaires n'aient fait une erreur en interprétant leur titre d'une manière trop littérale.)

Mon sentiment est que quand nous préparons un programme, cela peut être comme lorsque nous composons un poème ou un morceau de musique ; comme l'a dit Andrei Ershov [9], la programmation peut nous donner à la fois une satisfaction intellectuelle et une satisfaction émotionnelle, parce que cela consiste effectivement à maîtriser la complexité et à établir un système de règles consistantes.

En outre, quand nous lisons les programmes des autres, nous pouvons voir certains d'entre eux comme de véritables œuvres d'art d'ingéniosité. J'ai encore le souvenir du grand frisson que j'ai ressenti en lisant le listing du programme en assembleur de Stan Poley SOAP II en 1958 ; vous pensez probablement que je suis fou, et les styles ont probablement beaucoup changé depuis lors, mais à ce moment-là, cela avait beaucoup de signification pour moi que de voir à quel point un programme

système pouvait être élégant, spécialement en le comparant aux autres codes maladroits que j'avais trouvés dans d'autres listings que j'avais étudiés au même moment. La possibilité d'écrire de beaux programmes, même en assembleur, est ce qui m'a accroché à la programmation en premier lieu.

Certains programmes sont élégants, d'autres sont exquis, d'autres sont pétillants. Ce que je prétends, c'est qu'il est possible d'écrire de *grands* programmes, de *nobles* programmes, des programmes vraiment *magnifiques* !

Le goût et le style

L'idée d'un *style* de programmation devient d'avant-garde maintenant, et j'espère que la plupart d'entre vous avez vu l'excellent petit livre sur les *Éléments de style de programmation* de Kernighan et Plauger [16]. Dans cette relation faite ici, il est plus important de rappeler pour nous tous qu'il n'y a pas un style "meilleur" qu'un autre ; tout le monde a ses propres préférences, et c'est une erreur que d'essayer de forcer les gens à entrer dans un moule qui ne leur serait pas naturel. Nous entendons souvent dire "Je ne connais rien à l'art, mais je sais ce que j'aime". La chose importante est que vous *aimez* vraiment le style qui est le vôtre ; cela devrait être la façon par laquelle vous préférez vous exprimer.

Edsger Dijkstra a souligné ce point dans la préface de sa *Courte introduction à l'art de la programmation* [8] :

> Mon but est de transmettre l'important du bon goût et du style dans la programmation, [mais] les éléments spécifiques de style qui seront présentés ne serviront qu'à illustrer quels bénéfices peuvent être tirés du "style" en général. Selon ce point de vue, je me sens comme le professeur de composition au conservatoire : il n'apprend pas à ses élèves à composer une symphonie particulière, il les aide à trouver leur propre style et doit leur expliquer ce que cela implique. (C'est cette analogie qui m'a fait parler de "l'art de la programmation.")

Maintenant nous devons nous demander "Qu'est-ce qu'un bon style et qu'est-ce qu'un mauvais style ?" Nous ne devrions pas être trop rigides par rapport à cela en jugeant le travail d'autrui. Le philosophe du début du dix-neuvième siècle Jeremy Bentham a noté cela ainsi [3, Bk. 3, Ch. 1] :

> Les juges de l'élégance et du bon goût se considèrent comme des bienfaiteurs de l'humanité, alors qu'ils ne sont que les interrupteurs de son plaisir... Il n'y a pas de goût qui mérite l'épithète *bon*, à moins que ça ne soit un goût qui soit utilisé de telle manière que, par le plaisir qu'il procure vraiment, il procure également une utilité simultanée ou future : il n'y a pas de goût qui puisse mériter d'être qualifié de mauvais, à moins qu'il soit le goût pour une activité qui est d'une mauvaise tendance.

Quand nous appliquons nos propres préjugés pour "réformer" le goût de quelqu'un d'autre, nous pouvons inconsciemment lui dénier un plaisir totalement légitime. C'est pour cette raison que je ne condamne pas de nombreuses choses que les programmeurs font, même si je n'apprécierais jamais de les faire moi-même. La chose importante est qu'ils sont en train de créer quelque chose dont *ils*

ont le sentiment que c'est beau.

Dans le passage que je viens juste de citer, Bentham nous donne effectivement quelque avis à propos de certains principes d'esthétique qui sont meilleurs que d'autres, notamment l'"utilité" du résultat. Nous avons quelque liberté pour choisir nos propres standards de beauté, mais ce qui est particulièrement bien, c'est lorsque ce que nous trouvons nous-même comme beau est perçu par d'autres comme étant utile. Je dois concéder que j'apprécie vraiment d'écrire des programmes qui font le plus de bien, dans un certain sens.

Il y a de nombreux sens selon lesquels un programme peut-être "bon", bien sûr. En premier lieu, c'est particulièrement bon d'avoir un programme qui fonctionne correctement. Deuxièmement, c'est souvent bon d'avoir un programme qu'il ne sera pas trop compliqué de modifier, quand il faudra l'adapter à une nouvelle situation. Ces deux objectifs sont atteints lorsque le programme est à la fois facile à lire et compréhensible par une personne qui connaît le langage approprié.

Une autre façon importante pour un programme produit d'être bon, c'est qu'il soit capable d'interagir harmonieusement avec ses utilisateurs, spécialement en se comportant agréablement malgré les erreurs fournies en données par les utilisateurs. C'est vraiment un art de concevoir des messages d'erreur suffisamment significatifs ou d'anticiper des formats d'entrées suffisamment flexibles qui ne sont pas sources d'erreurs de la part de l'utilisateur.

Un autre aspect important de la qualité d'un programme est son efficacité à utiliser les ressources machine. Je suis désolé de dire que de nombreuses personnes de nos jours condamnent l'efficacité des programmes, en nous disant que c'est de mauvais goût de s'en préoccuper. La raison de cela est que nous sommes dans une époque de réaction par rapport à une époque récente dans laquelle l'efficacité était le seul critère d'évaluation des algorithmes, et les programmeurs dans le passé avaient tendance à être si préoccupés par l'efficacité qu'ils produisaient du code inutilement compliqué ; le résultat de cette complexité non-nécessaire a été que les préoccupations à propos de l'efficacité ont chuté, notamment à cause des difficultés de debugging et de maintenance de ces programmes si compliqués.

Le problème effectif est que les programmeurs ont dépensé trop de temps à s'interroger sur l'efficacité des programmes aux mauvais endroits et aux mauvais moments ; une optimisation prématurée est la source de tous les dangers (ou du moins de nombreux d'entre eux) en programmation.

Nous ne devrions pas nous perdre dans les détails et être globalement dispendieux, mais nous ne devrions pas non plus penser à l'efficacité uniquement en termes de tant de pourcent de gain ou perte sur un temps d'exécution, ou un espace utilisé. Quand nous achetons une voiture, beaucoup d'entre nous ne sommes pas trop regardant sur une différence de 50 ou 100 $ sur le prix, alors que nous serions parfois capables d'aller dans un magasin particulier pour acheter quelque chose à seulement 25 ¢ au lieu de 50 ¢. Mon idée par rapport à ce problème est qu'il y a un temps et un lieu pour l'efficacité ; j'ai discuté de ce sujet dans mon article sur la programmation structurée, qui paraît dans le numéro consultable en ce moment des *Computing Surveys* [21].

Moins de facilités : davantage de plaisir

Une chose plutôt curieuse que j'ai noté au sujet de la satisfaction esthétique est que notre plaisir est significativement augmenté quand nous réalisons quelque chose avec des outils limités. Par exemple, le programme dont je suis le plus content est un compilateur que j'ai un jour écrit pour un mini-ordinateur primitif qui avait seulement 4096 mots de mémoire, 16 bits par mot. Cela fait que l'on se sent vraiment être un virtuose lorsqu'on réussit à faire quelque chose dans des conditions si drastiques.

Un phénomène similaire a lieu dans beaucoup d'autres contextes. Par exemple, les gens tombent souvent amoureux de leur Volkswagen mais rarement de leur Lincoln Continental (qui marche vraisemblablement mieux). Quand j'ai appris à programmer, c'était un passe-temps populaire que de faire un maximum de choses avec des programmes qui tenaient sur une seule carte perforée. Je suppose que c'est le même phénomène qui fait savourer aux enthousiastes d'APL leurs programmes "tenant-en-une-ligne". Quand nous enseignons la programmation de nos jours, c'est un fait curieux que nous ne saisissions pas la passion d'un étudiant pour l'informatique jusqu'à ce qu'il ait également eu un cours avec les "mains dans le cambouis" de l'expérience de la programmation sur mini-ordinateur. L'utilisation de grosses machines avec leur système d'exploitation bizarre ne semble pas occasionner une quelconque passion pour la programmation, du moins pas au début.

La façon d'appliquer ce principe pour augmenter le plaisir que les programmeurs ont dans leur travail n'est pas évidente. Les programmeurs gémiraient sûrement si leur chef leur annonçait soudain que leur nouvel ordinateur aura seulement moitié moins de mémoire que l'ancien. Et je ne pense pas que l'on puisse attendre de quiconque, même pas du plus dévoué des "artistes programmeurs", qu'il accueille une telle éventualité de gaieté de cœur, parce que personne n'aime perdre ses acquis sans bonne raison. Un autre exemple peut aider à clarifier la situation : les réalisateurs ont fermement résisté à l'introduction du cinéma parlant dans les années 1920 parce qu'ils étaient fiers à juste titre de la manière dont ils pouvaient faire passer des messages dans leurs films muets. Un véritable artiste programmeur pourrait bien ressentir l'introduction de matériels plus puissants de la même façon ; les matériels de stockage de masse font pâlir nos méthodes de tri à l'ancienne. Mais aujourd'hui, les réalisateurs ne veulent plus revenir au cinéma muet, non pas parce qu'ils sont fainéants, mais parce qu'ils savent qu'il est parfaitement possible de faire de très beaux films en utilisant la technologie améliorée. La forme de leur art a changé, mais il reste toujours beaucoup de place pour l'art.

Comment ont-ils développé leur compétence ? Les meilleurs réalisateurs de films à travers les années semblent en général avoir appris leur art dans des circonstances relativement primitives, souvent dans d'autres pays avec une industrie cinématographique limitée. Et dans les dernières années, les choses les plus importantes que nous ayons apprises à propos de la programmation semblent avoir démarré avec des personnes qui n'avaient pas accès à des ordinateurs trop grands. La morale de cette histoire, il me semble, est que nous devrions utiliser l'idée des ressources limitées dans notre propre enseignement. Nous pouvons tous tirer du bénéfice du fait d'écrire à l'occasion des programmes "jouets", quand les restrictions artificielles sont levées, ainsi nous sommes forcés de pousser nos facultés à leur limite. Nous ne devrions pas vivre dans l'habitude constante du luxe, puisque cela a tendance à nous rendre léthargiques. L'art d'attaquer des mini-problèmes avec toute notre énergie aiguisera nos talents pour les problèmes réels, et l'expérience nous aidera à être plus

heureux encore lorsque nous réussirons sur des équipements moins restreints.

Dans une veine similaire, nous ne devrions pas avoir peur de faire de "l'art pour l'art" ; nous ne devrions pas nous sentir coupable à propos de programmes qu'on écrit juste pour le plaisir. J'ai vraiment pris mon pied en écrivant une fois un programme en ALGOL d'une seule instruction qui faisait appel à une procédure interne d'une telle façon qu'il calculait le m-ième nombre premier, plutôt que le produit intérieur [19]. Il y a quelques années, des étudiants de Stanford étaient excités d'avoir trouvé le programme FORTRAN le plus court qui s'imprimait lui-même, au sens où la sortie du programme était son propre code source. Le même problème a été considéré dans de nombreux autres langages. Je ne pense pas que c'était une perte de temps pour eux de travailler là-dessus ; et Jeremy Bentham, que j'ai déjà cité plus haut, n'aurait pas dénié l'"utilité" de telles utilisations du temps [3, Bk. 3, Ch. 1]. "Au contraire," écrit-il, "il n'y a rien dont l'utilité ne soit plus incontestable. À quoi l'utilité peut-elle être attribuée, si elle ne l'est pas à quelque chose qui est source de plaisir ?"

Fournir de beaux outils

Une autre caractéristique de l'art moderne est l'insistance qu'elle place dans la créativité. Il semblerait que de nombreux artistes de nos jours ne se soucient de rien moins que de créer de belles choses ; c'est seulement la nouveauté d'une idée qui importe. Je ne considère pas que la programmation des ordinateurs pourrait ressembler à l'art moderne selon ce sens-là, mais cela m'amène à une observation qui me semble importante. Parfois nous sommes contraints d'effectuer une tâche de programmation qui est désespérément terne, ne nous permettant aucune échapattoire créative ; et à ce moment-là, quelqu'un pourrait très bien me dire, "alors, c'est beau la programmation ? C'est bien beau de crier que je devrais prendre un grand plaisir à créer des progammes élégants et charmants, mais comment suis-je supposé transformer tout ce bazar en travail d'art ?" Bon, c'est vrai, toutes les tâches de programmation ne vont pas forcément être amusantes. Considérons la "ménagère" qui doit chaque jour nettoyer la même table : il n'y a pas forcément de place pour la créativité ou l'art dans toute situation. Mais même dans de tels cas, il y a moyen de faire une grosse amélioration : c'est toujours un plaisir de faire des travaux de routine si vous avez de beaux outils avec lesquels les faire. Par exemple, une personne aura vraiment plaisir à nettoyer la table, jour après jour, si c'est une table merveilleusement bien ouvragée avec un bois de luxe.

Donc je veux adresser mes remarques terminales aux programmeurs systèmes et aux concepteurs de machines qui produisent les outils avec lesquels le reste d'entre nous travaillons. *S'il vous plaît*, donnez-nous des outils qui sont agréables à utiliser, spécialement pour nos travaux de routine, au lieu de fournir quelque chose avec quoi nous devons nous battre. S'il vous plaît, donnez-nous des outils qui nous encouragent à écrire de meilleurs programmes, en augmentant notre plaisir quand nous le faisons.

C'est très difficile pour moi de convaincre de jeunes gens que la programmation est belle, quand la première chose que je dois leur dire est comment taper "slash slash JOB égal etc, etc." Même les langages pour le travail de contrôle peuvent être conçus de telle façon que c'en est un plaisir que de les utiliser, au lieu d'être strictement fonctionnel.

Les concepteurs de matériels informatiques peuvent faire que leurs machines soient beaucoup plus

agréables à utiliser, par exemple en fournissant une arithmétique des réels qui satisfasse des lois mathématiques simples. Les possibilités offertes par la plupart des machines actuellement en vente rendent le travail d'analyse rigoureuse des erreurs difficile et sans espoir, mais les opérations proprement conçues devraient encourager les analystes numériques à fournir de meilleures sous-procédures dont la précision a été certifiée (cf. [20, p. 204]).

Considérons aussi ce que peuvent faire les concepteurs de logiciels. La meilleure manière de rester dans l'esprit d'un utilisateur système est de lui fournir des procédures avec lesquelles il peut interagir. Nous ne devrions pas rendre les systèmes trop automatiques, de telle façon que l'action part toujours derrière la scène ; nous devons offrir à l'utilisateur programmeur une possibilité de diriger sa créativité vers des canaux utiles. Une chose que tous les programmeurs ont en commun est qu'ils apprécient de travailler avec des machines ; alors gardons-les dans la boucle. Certaines tâches sont mieux faites par la machine alors que d'autres sont mieux faites par les humains ; et un système convenablement spécifié trouvera l'équilibre correct. (J'ai essayé d'éviter une automatisation maldirigée pendant de nombreuses années, cf. [18].)

Les outils d'évaluation des programmes sont un bon cas d'école. Pendant des années, les programmeurs ne savaient pas comment étaient effectivement distribués les coûts de calcul dans leurs programmes. L'expérience montre que presque tous les programmeurs ont une mauvaise idée de l'endroit effectif où se situent les nœuds d'engorgement de leurs programmes ; il n'est pas étonnant que les tentatives pour tendre vers plus d'efficacité se trompent, si on ne donne jamais au programmeur une estimation de la baisse des coûts selon les lignes de code qu'il a écrites. Sa tâche ressemble à celle d'un couple de jeunes mariés qui essaient de programmer un budget équilibré sans connaître le coût des items individuels comme la nourriture, le logement, l'habillement. Tout ce que nous avons fourni aux programmeurs, c'est un compilateur optimisé, qui fait mystérieusement quelque chose aux programmes qu'il traduit, mais qui n'explique jamais ce qu'il fait. Heureusement, nous voyons apparaître des systèmes qui accordent à l'utilisateur un peu de crédit pour son intelligence ; ils fournissent automatiquement une instrumentation des programmes et un retour à propos des coûts effectifs. Ces systèmes expérimentaux ont rencontré un franc succès, parce qu'ils produisent des améliorations mesurables, et spécialement aussi parce qu'ils sont marrants à utiliser, donc j'ai confiance sur le fait que ce n'est qu'une affaire de temps avant que l'utilisation de tels systèmes devienne une manière standard de procéder. Mon article dans *Computing Surveys* [21] discute de cela plus avant, et présente quelques idées d'autres manières selon lesquelles une procédure interactive peut améliorer la satisfaction des programmeurs utilisateurs.

Les concepteurs de langage aussi ont une obligation de fournir des langages qui encouragent un bon style, dans la mesure où nous savons tous combien le style est fortement influencé par le langage dans lequel il s'exprime. Le sursaut d'intérêt actuel pour la programmation structurée a révélé qu'aucun de nos langages existant n'est vraiment idéal pour gérer la structure des programmes et des données, et on ne sait pas encore de façon claire comment devrait être un tel langage idéal. Par conséquent, je regarderai attentivement les expériences de conception des langages qui seront faites dans les quelques prochaines années.

Résumé

Pour résumer : nous avons vu que la programmation des ordinateurs est un art, parce qu'elle applique des connaissances accumulées au monde, parce qu'elle requiert des compétences et de l'ingéniosité, et particulièrement parce qu'elle produit des objets de beauté. Un programmeur qui se voit inconsciemment lui-même comme un artiste appréciera davantage ce qu'il fait et le fera mieux. Ainsi, nous pouvons nous réjouir du fait que les personnes qui donnent des conférences d'informatique parlent de l'*état de l'Art*.

Références

1. Bailey, Nathan. *The Universal Etymological English Dictionary*, T. Cox, London, 1727, See "Art," "Liberal," and "Science."

2. Bauer, Walter F., Juncosa, Mario L., and Perlis, Alan J. ACM publication policies and plans. *J. ACM* 6 (Apr. 1959), 121-122.

3. Bentham, Jeremy. *The Rationale of Reward*. Trans. from *Théorie des peines et des récompenses*, 1811, by Richard Smith. J. & H. L. Hunt, London, 1825.

4. *The Century Dictionary and Cyclopedia 1.* The Century Co., New York, 1889.

5. Clementi, Muzio. *The Art of Playing the Piano.* Trans. from *L'art de jouer le pianoforte* by Max Vogrich. Schirmer, New York, 1898.

6. Colvin, Sidney. "Art." *Encyclopaedia Britannica*, eds 9, 11, 12, 13, 1875-1926.

7. Coxeter, H. S. M. Convocation address, Proc. 4th Canadian Math. Congress, 1957, pp. 8-10.

8. Dijkstra, Edsger W. EWD316 : *A Short Introduction to the Art of Programming.* T. H. Eindhoven, The Netherlands, Aug. 1971.

9. Ershov, A. P. Aesthetics and the human factor in programming, *Comm. ACM 15* (July 1972), 501-505.

10. Fielden, Thomas. *The Science of Pianoforte Technique*. Macmillan, London, 1927.

11. Gore, George, *The Art of Scientific Discovery*. Longmans, Green, London, 1878.

12, Hamilton, Willian, *Lectures on Logic 1.* Wm. Blackwood, Edinburgh, 1874.

13. Hodges, John A. Elementary Photography : *The "Amateur Photographer" Library 7.* London, 1893. Sixth ed, revised and enlarged, 1907, p. 58.

14. Howard, C. Frusher. Howard's *Art of Computation* and golden rule for equation of payments for schools, business colleges and self-culture.... C.F. Howard, San Francisco, 1879.

15. Hummel, J.N. *The Art of Playing she Piano Forte*. Boosey, London, 1827.

16. Kernighan B.W., and Plauger, P.J. *The Elements of Programming Style*. McGraw-Hill, New York, 1974.

17. Kirwan, Richard. *Elements of Mineralogy*. Elmsly, London, 1784.

18. Knuth, Donald E, Minimizing drum latency time. *J. ACM 8* (Apr. 1961), 119-150.

19. Knuth, Donald E., and Merner, J.N. ALGOL 60 confidential. *Comm. ACM 4* (June 1961), 268-272.

20. Knuth, Donald E. Seminumerical Algorithms : *The Art of Computer Programming 2*. Addison-Wesley, Reading, Mass., 1969.

21. Knuth, Donald E. Structured programming with go to statements, *Computing Surveys 6* (Dec. 1974), pages in makeup.

22. Kochevitsky, George. *The Art of Plano Playing : A Scientific Approach*. Summy-Birchard, Evanston, III, 1967.

23. Lehmer, Emma. Number theory on the SWAC. *Proc. Symp. Applied Math. 6*, Amer. Math. Soc. (1956), 103-108.

24. Mahesh Yogi, Maharishi. *The Science of Being and Art of Living*. Allen & Unwin, London, 1963.

25. Malevinsky, Moses L. *The Science of Playwriting*. Brentano's, New York, 1925.

26. Manna, Zohar, and Pnueli, Amir. Formalization of properties of functional programs. *J. ACM 17* (July 1970), 555-569.

27. Marckwardt, Albert H. Preface to *Funk and Wagnall's Standard College Dictionary*. Harcourt, Brace & World, New York, 1963, vii.

28. Mill, John Stuart. *A System af Logic, Ratiocinative and Inductive*. London, 1843, The quotations are from the introduction, §2, and from Book 6, Chap. 11 (12 in later editions), §5.

29, Mueller, Robert E. *The Science of Art*. John Day, New York, 1967.

30. Parsons, Albert Ross. *The Science of Pianoforte Practice*. Schirmer, New York, 1886.

31. Pedoe, Daniel. *The Gentle Art of Machematics*. English U. Press, London, 1953.

32. Ruskin, John. *The Stones of Venice 3*. London, 1853.

33. Salton, G.A. Personal communication, June 21, 1974.

34. Snow, C.P. *The two cultures. The New Statesman and Nation 52* (Oct. 6, 1956), 413-414.

35. Snow, C.P. *The Two Cultures : and a Second Look*. Cambridge University Press, 1964.