Pensées, proverbes et citations

J-P. Rosen

Voici quelques réflexions personnelles ou qui font partie de la sagesse populaire informatique. Lorsque je le connaissais, j'ai donné le nom de l'auteur. Les dictons suivis de [?] sont d'auteurs inconnus (de moi); s'ils se reconnaissent, je serais très heureux de leur rendre leur dû. Les autres sont de mon crû.

On pourra trouver certains de ces "proverbes" partisans: ils le sont effectivement. Rien ne vous oblige à les approuver...

I Généralités

L'ordinateur ne crée pas de fonctionnalité nouvelle, il ne fait qu'amplifier la puissance pré-existante du cerveau humain; c'est ainsi que grâce à l'informatique, une erreur d'une seule personne peut avoir des conséquences catastrophiques sur des millions d'autres.

Si les ordinateurs accomplissent tellement plus de travail que les être humains, c'est qu'ils ne doivent pas s'interrompre toutes les cinq minutes pour répondre au téléphone. [?]

II A propos de systèmes d'exploitation

Chaque fois que je travaille sous UNIX, j'ai l'impression qu'un génie malfaisant est perché sur mon épaule, qui attend la première faute de frappe pour massacrer tous mes fichiers.

Tout système dont la fiabilité repose sur UNIX est non fiable.

Un mauvais standard vaut mieux qu'un dispositif excellent mais non standardisé, et l'IBM/PC est là pour le démontrer.

Les sémaphores sont à la programmation parallèle ce que le GOTO est à la programmation séquentielle.

Lorsque vous envoyez un message par UUCP, doublez vos chances qu'il arrive à bon port! Mettez-le AUSSI dans une bouteille.

III A propos de génie logiciel

Je n'ai qu'une petite tête, et il faut bien que je vive avec [Hoare].

Le nombre de touches frappées pour faire marcher un programme mal écrit peut être supérieur à celui nécessaire pour le réécrire proprement. Mais on ne va pas jeter ce qu'on a déjà écrit...

Rien n'est plus difficile que de faire simple.

... diviser chacune des difficultés que j'examinerais en autant de parties qu'il se pourrait et qu'il serait requis pour les mieux résoudre. [Descartes]

L'on doit bien observer qu'il n'y a pas de chose plus difficile à manier, ni dont le succès soit plus douteux, et qu'on fasse avec plus de danger que d'agir en chef pour introduire des institutions nouvelles. [Machiavel]

On ne construit pas un pont sur un estuaire en extrapolant une passerelle sur une rivière. [?]

L'homme vit comme s'il ne devait jamais mourir; le programmeur écrit comme si ses programmes ne pouvaient avoir de bugs.

Le jour où un programmeur devient indispensable à une entreprise, il faut le mettre à la porte (et si possible, la veille).

Il ne faut jamais tenter de faire marcher un programme contre sa volonté. S'il résiste, c'est qu'il y a un problème de conception.

La documentation est comme l'huile de foie de morue: personne n'aime ça, mais cela doit bien servir à quelque chose puisque les supérieurs y attachent tellement d'importance. [?]

Un bon programme ne comporte pas de commentaire algorithmique. Un commentaire dans un algorithme est un aveu que le code n'est pas écrit de façon suffisamment lisible.

Toute tentative de programmer uniquement en fonction de l'efficacité ne sert qu'à démontrer une chose: le programme ne perd jamais du temps là où on le croyait a priori.

Proposez à Renault un dispositif, amortissable en 10 ans, qui améliore de 5% la productivité des chaînes et l'on vous recevra comme un sauveur. Proposez un outil de génie logiciel, amortissable en 3 mois, qui double la productivité des programmeurs on vous dira que c'est trop cher.

Le logiciel est la seule branche de l'industrie où l'on trouve normal de supprimer les dispositifs de sécurité pour améliorer les performances.

Efficacité, que de bugs on commet en ton nom!

Tout problème de codage est un problème de conception qui s'ignore.

Les hackers me font penser aux navigateurs solitaires: ils sont extrèmement doués, mais personne n'ose partir avec eux dans de telles conditions de sécurité.

Si les programmeurs ...

... respectaient les normes de programmation
... écrivaient de la documentation
... testaient soigneusement les programmes
... maitrisaient aisément les relations complexes
... ne s'appuyaient que sur les propriétés démontrables des programmes

Ada serait inutile!

IV A propos de langages de programmation

Ce qui est important dans un langage de programmation, ce n'est pas ce qu'il autorise, c'est ce qu'il interdit.

Langage sans méthode n'est que ruine de la société de service.

Les étudiants qui ont été en contact avec BASIC sont mentalement mutilés au-delà de tout espoir de régénération [Dijkstra].

Il est faux de dire que BASIC est le plus mauvais des langages; il ne faut pas oublier dBaseIII...

L'assembleur est un langage de trop haut niveau pour la programmation temps-réel [Les programmeurs temps-réel].

Si l'on en juge d'après le parc de machines sur lesquelles il fonctionne, le langage le plus portable est l'assembleur 370.

Le langage C est fondé sur l'hypothèse que les programmeurs sont des gens responsables et qui savent ce qu'ils font. Ada est fondé sur l'hypothèse que les programmeurs sont des êtres humains.

C est un langage portable sur machines 16 bits, fonctionnant sous Unix, qui adressent directement l'octet, dont l'espace d'adressage des différents types de données et de programmes est unique, qui travaillent en ASCII codé sur 8 bits, dont la taille d'un pointeur est égale à celle d'un entier, et à la condition de prendre certaines précautions au codage...

Je n'aime pas APL; je préfère écrire mes programmes en anglais qu'en grec.

Si Ada s'était appelé SUPERFORTRAN, toute la communauté scientifique l'utiliserait déjà.

Un bon programmeur FORTRAN peut écrire du FORTRAN dans n'importe quel langage [?].

Si l'on pouvait faire en sorte que les programmeurs écrivent les programmes en langue naturelle, on s'apercevrait qu'ils ne savent pas exprimer leurs idées en langue naturelle. [?]

On peut écrire très proprement dans n'importe quel langage (y compris C); on peut écrire salement dans n'importe quel langage (y compris Ada); mais Ada est le seul langage où il soit plus fatigant d'écrire salement que proprement.

Dans les laboratoires, les chercheurs passent leur vie à expliquer que les appareils achetés l'année dernière sont devenus obsolétes et qu'il faut absolument se procurer le dernier modèle; mais ils programment toujours au moyen d'un langage vieux de 30 ans.

Lim(Modula-N) = Ada [R. Pattis]
N => infini

Les "astuces de programmation" sont une calamité. Un bon programme dit ce qu'il fait et fait ce qu'il dit.

La programmation souffre d'un gros défaut: c'est terriblement amusant. Il est grand temps que les programmeurs arrêtent de s'amuser pour songer à faire de la production industrielle de logiciels.

Programmer en Ada, c'est remplacer le plaisir du jeu par la satisfaction du travail bien fait.

Ceux qui se sont mis à Ada passent leur temps à énumérer leurs récriminations, mais pas un ne retournerait à son ancien langage.

Reconnaissons-le: Ada n'est pas parfait. C'est *seulement* le meilleur des langages (séquentiels procéduraux) existants.

Dans les défauts reprochés à Ada, j'ai lu une fois [communications of the ACM] qu'il était inapproprié pour l'enseignement de l'informatique dans les écoles primaires. Dont acte, mais ça ne faisait pas vraiment partie du cahier des charges...

Avec l'émergence des langages parallèles (dont Ada bien sûr), il ne faudra plus se demander si l'on est forcé d'utiliser le parallélisme, mais quels sont les algorithmes que l'on est contraint d'exécuter en séquentiel.

Les étapes du passage à Ada:

1) FORTRAN me va tout à fait, pourquoi chercher autre-chose
2) Il y a quelques fonctionnalités intéressantes...
3) Pourquoi les gens ne passent-ils pas tous à Ada?

Avec Ada, le compilateur n'est plus un esclave qui traduit votre programme en assembleur, mais un aide qui en vérifie la cohérence logique.

V Lois de Murphy

Les "lois" suivantes sont extraites, parfois avec des détournements personnels, des lois parues dans *Murphy's Law and other reasons why things go wrong* [A. Bloch].

V.1.1.1 Théorème de Stockmayer

Si ça a l'air facile, c'est difficile. Si ça a l'air difficile, c'est carrément impossible. Si ça a l'air impossible, c'est un compilateur Ada.

V.1.1.2 Loi de Booker

Dix grammes d'abstraction valent des tonnes de bricolage.

V.1.1.3 Loi de Klipstein

Les défauts n'apparaissent qu'après que le programme ait passé (avec succès) la phase d'intégration.

V.1.1.4 Loi d'Osborn

(Malheureusement intraduisible): Variables won't; Constants aren't.

V.1.1.5 Lois de Gilb

Les ordinateurs ne sont pas fiables; mais les êtres humains sont encore moins fiables.

Les investissements en fiabilité augmenteront jusqu'à dépasser le coût probable des erreurs, où jusqu'à ce quelqu'un insiste pour qu'on se mette à faire du travail utile.

Les investissements en fiabilité des programmes en C augmenteront jusqu'à ce que le programmeur en ait assez et trouve plus intéressant de se mettre à écrire un super-utilitaire pour UNIX, où jusqu'à ce que quelqu'un insiste pour tout recoder en Ada.

V.1.1.6 Loi de Brook

Doubler le nombre de programmeurs sur un projet en retard ne fait que doubler le retard.

V.1.1.7 Loi de Lubarsky

Il y a toujours un bug de plus.

V.1.1.8 Loi de Patton

Un bon langage aujourd'hui vaut mieux qu'un langage parfait demain.

V.1.1.9 Loi de Blaauws

Une technologie établie tend à persister malgré l'apparition de nouvelles technologies. (Le lecteur traduira de lui-même en termes de langages de programmation...).

V.1.1.10 Seconde loi de Weinberg

Si les architectes construisaient les maisons comme les programmeurs écrivent les programmes, le premier picvert venu détruirait la civilisation.

VI Petit glossaire

Cauchemar:	Porter un programme en C sur une machine 36 bits.
Dieu:	Etre mythique qu'on ne rencontre jamais. Syn: documentation.
Editeur de liens:	Processeur indispensable dont personne ne sait ce qu'il fait.
Efficacité:	Excuse utilisée par les hackers pour justifier n'importe quelle horreur.
OS/2:	Moitié d'un système d'exploitation. [?]
Paradoxe:	C++ est entièrement compatible avec C, avec le typage fort en plus.
Validation:	Ensemble de tests permettant de prouver que le programme est capable de passer les tests.