Traduction d'une interview de

Donald E. Knuth

par

Andrew Binstock

25 avril 2008

Journal numérique Informit

http://www.informit.com/articles/article.aspx?p=1193856

Andrew Binstock et Donald Knuth [1] discutent du succès de l'open source, du problème de l'architecture multi-cœur, du manque désappointant d'intérêt pour la programmation littéraire, de la menace du code réutilisable, et de cette légende urbaine sur le fait de gagner un concours de programmation avec une seule compilation.

Andrew Binstock : Vous êtes un des pères de la révolution de l'open-source, même si l'on ne vous annonce pas comme tel. Vous avez affirmé précédemment que vous avez libéré Teχ comme open-source à cause du problème des implémentations propriétaires en ce moment, et pour inviter à ce que le code soit corrigé, ces deux points étant des facteurs-clefs des projets open-source actuels. Avez-vous été surpris par le succès de l'open-source depuis ce moment ?

Knuth : Le succès du code open-source est peut-être la seule chose qui ne m'ait pas surpris dans le domaine informatique durant les quelques dernières décennies. Mais il n'a pas encore atteint son plein potentiel ; je crois que les programmes open-source commenceront à être complètement dominants lorsque l'économie ira de plus en plus des produits vers les services, et lorsque de plus en plus de volontaires réussiront à améliorer le code.

Par exemple, un code open-source peut produire des milliers d'exécutables, parfaitement adaptés aux configurations des utilisateurs individuels, alors qu'un logiciel commercial existe en général seulement en quelques versions. Un exécutable binaire générique doit inclure des choses comme ces instructions "sync" inefficaces qui sont complètement inappropriées pour la plupart des installations ; un tel gaspillage est supprimé lorsque le code source est hautement configurable. Cela devrait être un énorme atout pour l'open-source.

Je pense également que quelques programmes, comme Adobe Photoshop, seront toujours de qualité supérieure à leurs compétiteurs comme TheGimp pour quelques raisons, je ne sais franchement pas pourquoi ! Je souhaite payer un bon argent pour du vraiment bon logiciel, si je crois qu'il a été produit par les meilleurs programmeurs.

1. Donald Knuth est entre autres l'auteur de *L'Art de la programmation des ordinateurs*, Volume 4, Fascicule 0 : Introduction aux algorithmes combinatoires et aux fonctions booléennes, livre qui vient d'être publié.

Rappelez-vous cependant, que mon opinion sur les questions économiques est profondément suspecte, puisque je suis juste un enseignant et un scientifique. Je ne comprends quasiment rien aux lois du marché.

Andrew : Une histoire raconte que vous avez participé à un concours de programmation à Stanford (je crois) et que vous avez soumis le programme qui a gagné correctement après une seule compilation. Est-ce vrai ? Dans cette veine, les développeurs d'aujourd'hui construisent fréquemment leurs programmes en écrivant des petits incréments de code, suivi d'une compilation immédiate et la création et l'exécution d'unités de tests. Quel est votre sentiment par rapport à cette approche du développement de logiciels ?

Knuth : L'histoire que vous avez entendue est typique des légendes qui sont basées sur seulement un petit noyau de vérité. Voici ce qui s'est réellement passé : John McCarthy a décidé en 1971 d'avoir une journée mémoire consacrée à une course de programmation. Tous les concurrents sauf moi travaillaient à son Laboratoire d'IA là-haut dans les collines au-dessus de Stanford, en utilisant le système de temps partagé WAITS ; j'étais en bas sur le campus principal, où le seul ordinateur que je pouvais utiliser était un mainframe dans lequel je devais mettre des cartes et les soumettre à la compilation en mode batch. Je programmais en ALGOL W (le langage de Wirth, le prédécesseur de Pascal). Mon programme n'a pas marché du premier coup, mais par chance, j'ai pu utiliser l'excellent système de debugging d'Ed Satterthwaite pour ALGOL W, du coup, je n'ai eu besoin que de deux exécutions. Pendant ce temps, les types qui utilisaient WAITS ne pouvaient pas obtenir assez de cycles machine parce que leur machine était complètement surchargée (je pense que celui qui est arrivé deuxième, en utilisant cette approche "moderne", a terminé une heure environ après que j'ai soumis le programme gagnant avec des méthodes à l'ancienne). Ce n'était pas un concours très fair-play.

Quant à votre véritable question, l'idée d'une compilation immédiate et de "tests unitaires", je n'y fais appel que rarement, quand je cherche mon chemin dans un environnement totalement inconnu et que j'ai besoin d'avoir un retour sur ce qui marche et ce qui ne marche pas. Sinon, beaucoup de temps est gaspillé à des activités que je n'aurais jamais nécessitées ou même auxquelles je n'aurais jamais pensé. "On ne doit se moquer de rien".

ANDREW : Un des problèmes émergents pour les développeurs, spécialement les développeurs côté client, est de changer leur manière d'écrire les programmes en terme de *thread* (fil d'exécution). Ce problème, amené par l'arrivée de PC multi-cœurs peu chers, nécessitera sûrement que de nombreux algorithmes soient réécrits pour un matériel à architecture parallèle, ou du moins pour être viable sur un tel matériel. Jusque là, la plupart du travail que vous avez publié pour le Volume 4 de *L'Art de la programmation des ordinateurs (TAOCP)* ne semble pas prendre en compte cette dimension. Avez-vous l'intention d'attaquer des problèmes de programmation concurrente et parallèle dans des travaux à venir, spécialement dans la mesure où cela semble être particulièrement adapté pour les problèmes combinatoires sur lesquels vous travaillez en ce moment ?

KNUTH : Le domaine des algorithmes combinatoires est si vaste que j'aurai de la chance si je réussis à compresser ses aspects séquentiels dans trois ou quatre volumes, et je ne pense pas que les méthodes séquentielles seront sans importance. Inversement, la demi-vie des techniques parallèles est très courte, parce que le matériel (hardware) change rapidement et chaque nouvelle machine nécessite une approche un peu différente. Du coup, j'ai décidé il y a longtemps que je collerais à ce que je connais le mieux. Les autres personnes comprennent les machines parallèles bien mieux que je ne le fais ; les programmeurs devraient les écouter, plutôt que moi, pour être guidés dans tout ce qui a trait à la simultanéité.

ANDREW : Les vendeurs de processeurs multi-cœurs ont exprimé de la frustration face à la difficulté d'amener les développeurs à ce modèle. Comme professeur, que pensez-vous de cette transition et comment la faire advenir ? Est-ce une question d'outils propres, tels qu'un meilleur support natif de la concurrence dans les langages, ou de modèles d'exécution ? Ou y a-t-il d'autres solutions ?

KNUTH : Je ne veux pas descendre complètement votre question. Je pourrais plutôt flamber un peu à propos de mon mécontentement personnel par rapport à la tendance actuelle vers l'architecture multi-cœurs. Pour moi, c'est plus ou moins comme si les concepteurs d'hardware étaient à cours d'idées, et qu'ils essayaient de rejeter la faute de la prochaine disparition de la loi de Moore sur les programmeurs en nous donnant des machines qui fonctionnent

plus vite seulement sur quelques benchmarks clefs ! Je ne serais pas surpris du tout si l'idée complète du parallélisme finissait par être un flop, pire que l'approche "Itanium" qui était supposée être si terrible, jusqu'à ce qu'on réalise que les compilateurs qu'elle nécessitait étaient complètement impossibles à écrire.

Laissez-moi le dire autrement : durant les 50 dernières années, j'ai écrit bien plus d'un millier de programmes, la plupart d'entre eux de taille substantielle. Je ne peux même pas penser à cinq de ces programmes, qui auraient été améliorés de façon notable par le parallélisme et le multi-threading. De façon certaine, par exemple, les processeurs multiples ne sont d'aucune utilité pour Teχ. Mon collègue Kunle Olukotun note que, si l'usage de Teχ devenait un goulot d'étranglement, de manière que les personnes aient des douzaines de processeurs et aient vraiment besoin d'augmenter leur vitesse de composition de manière terrible, une version de Teχ pourrait être développée qui utilise la "spéculation" pour composer une douzaine de chapitres simultanément : chaque chapitre pourrait être composé sous la condition que les chapitres précédents ne feraient rien d'étrange qui dérangerait la logique par défaut. Si cette supposition n'était pas respectée, on pourrait revenir à la méthode normale de faire les chapitres un par un ; mais dans la majorité des cas, quand seule une composition normale est utilisée, le processus aurait plutôt été 12 fois plus rapide. Les personnes qui sont préoccupées par la vitesse pourrait adapter leur comportement et utiliser Teχ d'une manière disciplinée..

Combien de programmeurs connaissez-vous qui sont enthousiastes à propos de ces machines qui nous sont promises dans le futur ? Je n'ai entendu que des griefs de la part des personnes du logiciel, bien que les types du hardware dans notre département m'assurent que j'ai tort.

Je sais que d'importantes applications existent pour le parallélisme, telles que le graphisme, le cassage de code, la reconnaissance d'images, la simulation de processus physiques et biologiques, etc. Mais toutes ces applications nécessitent un code dédié et des techniques spécifiques, qui devront changer substantiellement toutes les quelques années.

Même si j'en sais assez à propos de ces méthodes pour écrire à leur propos dans TAOCP, mon temps serait largement gaspillé, parce que très vite, il

n'y aurait plus de raison pour personne de lire les parties du livre en question. (De façon similaire, en préparant la troisième édition du Volume 3, j'ai programmé de virer la plupart du matériau concernant le tri de cassettes magnétiques. Ce matériau a été un jour le sujet le plus chaud de tout le domaine logiciel, mais maintenant, cela gaspille du papier à l'impression).

La machine que j'utilise aujourd'hui a des processeurs duaux. J'ai l'habitude de les utiliser tous les deux seulement quand je lance deux travaux indépendants en même temps ; c'est chouette, mais ça arrive seulement quelques minutes par semaine. Si j'avais quatre processeurs, ou huit, ou plus, je continuerais de ne pas faire mieux, en considérant le genre de travail que je fais même si j'utilise un ordinateur chaque jour pendant la plus grande partie de la journée. Du coup, pourquoi devrais-je être si content du futur que les vendeurs de matériel promettent ? Ils pensent à une bulle magique qui viendra grâce à ses multi-processeurs améliorer mon type de travail ; je pense que c'est un rêve creux (Non, ça n'est pas la bonne métaphore ! Les "pipelines" travaillent vraiment pour moi, mais pas les threads (les fils d'exécution). Peut-être que le mot que je veux, c'est "bulles").

Du point de vue opposé, je remercie du fait que la navigation sur le web sera probablement accélérée par les multi-cœeurs. J'ai parlé de mon travail technique, pourtant, pas de récréation. J'admets aussi que je n'ai pas eu beaucoup de brillantes idées à propos de ce que je souhaiterais que les concepteurs de matériel fournissent plutôt que des processeurs multi-cœurs, maintenant qu'ils ont commencé à casser un mur par rapport au calcul séquentiel (mais ma conception de MMIX contient quelques idées sur ce qui améliorerait substantiellement la performance actuelle des programmes qui me concernent davantage... au coût de l'incompatibilité avec les programmes x86).

Andrew : L'un de quelques-uns de vos projets qui n'ont pas été suivis par beaucoup de monde est la programmation littéraire. Quelles sont vos idées sur ce fait que la programmation littéraire n'ait pas trop marché ? Et y a-t-il quelque-chose que rétrospectivement vous auriez fait autrement à propos de la programmation littéraire ?

Knuth : La programmation littéraire est une chose très personnelle. Je pense que ça pourrait être terrible, mais c'est vraisemblablement dû au fait que je suis une très étrange personne. Elle a des dizaines de milliers de fans,

mais pas des millions.

De mon expérience, du logiciel créé par la programmation littéraire s'est avéré être significativement meilleur que du logiciel développé par des méthodes plus traditionnelles. Déjà quand le programme est ok, je lui donne une note de C (ou peut-être C++), mais pas F ; de ce fait, les méthodes traditionnelles restent avec nous. Parce qu'elles sont comprises par une vaste communauté de programmeurs, la plupart des gens n'ont pas une grosse incitation à changer, de la même façon que je ne suis pas très motivé par le fait d'apprendre l'Esperanto, même si ce serait préférable de l'apprendre plutôt que d'apprendre l'anglais et l'allemand et le français et le russe (à permutation près).

Jon Bentley s'est probablement frappé la tête quand un jour on lui a demandé pourquoi la programmation littéraire ne s'était pas étendue au monde entier comme une tempête. Il a observé qu'un petit pourcentage de la population mondiale programme bien, et qu'un petit pourcentage de la population écrit bien ; apparemment, je demande à tous d'appartenir aux deux sous-ensembles.

Toujours pour moi, la programmation littéraire est certainement la chose la plus importante qui est sortie du projet Teχ. Pas seulement parce qu'elle me rend capable d'écrire et maintenir les programmes plus vite et de façon plus fiable que ça n'avait jamais été le cas auparavant, et parce qu'une des plus grandes sources de joie depuis les années 80 a vraiment été de temps en temps indispensable. Quelques-uns de mes plus importants programmes, comme le méta-simulateur MMIX, n'aurait pas pu être écrit avec n'importe quelle autre méthodologie dont j'ai jamais entendu parler. La complexité était simplement trop intimidante pour mon cerveau limité pour que je puisse l'attaquer autrement qu'avec la programmation littéraire ; sans ce paradigme, l'entreprise complète aurait échoué misérablement.

Si les gens découvrent de bonnes manières d'utiliser les machines multi-threads qui sont tendance en ce moment, je m'attendrais à ce que cette découverte provienne de personnes qui utilisent régulièrement la programmation littéraire. La programmation littéraire est ce dont vous avez besoin pour vous élever au-dessus du niveau ordinaire de réussite. Mais je ne crois pas que l'on puisse obliger quiconque à quoi que ce soit. Si la programmation

littéraire n'est pas votre style, s'il vous plaît, oubliez-la et faites ce que vous voulez. Si personne d'autre que moi ne l'aime, laissons la mourir.

De façon positive, j'ai été content d'apprendre que les conventions du CWEB sont presque des équipements standards sans logiciels préinstallés comme des Makefiles, quand j'obtiens des Linux sur l'étagère actuellement.

ANDREW : Dans le Fascicule 1 du Volume 1, vous avez réintroduit le calculateur MMIX, qui est la mise-à-jour 64-bits de la vénérable machine MIX que les étudiants en informatique ont été amenés à connaître depuis de nombreuses années. Vous avez précédemment décrit MMIX en grands détails dans MMIXware. J'ai lu des parties de ces deux livres, mais ne peux dire si le Fascicule met à jour ou modifie ce qui était dans MMIXware, ou bien si c'est un pur synopsis. Pourriez-vous clarifier ?

KNUTH : Le Volume 1 Fascicule 1 est une introduction pour le programmeur, qui inclut des exercices instructifs et ce genre de choses. Le livre MMIXware est un manuel de référence détaillé, quelque-chose de succinct et aride, plus un bouquet de programmes littéraires qui décrivent le prototype logiciel pour les gens qui travailleront au-dessus de lui. Les deux livres définissent le même ordinateur (une fois que les erreurs de MMIXware seront corrigées à partir de la version de mon site web). Pour la plupart des lecteurs de TAOCP, le premier fascicule contient tout ce dont ils auront un jour besoin ou qu'ils souhaitent savoir à propos de MMIX.

Je voudrais noter aussi, cependant, que MMIX n'est pas une seule machine ; c'est une architecture avec presque toutes les variétés sans limitation d'implémentations, dépendant des différents choix d'unités fonctionnelles, des différentes configurations pipelines, des différentes approches du problème des instructions multiples, des différentes manières d'effectuer la prédiction de branchements, des différents tailles de cache, des différentes stratégies de remplacement du cache, des différentes vitesses des bus, etc. Quelques instructions et/ou registres peuvent être émulés par du logiciel sur des versions "moins chères" du matériel. Et etc. C'est un lit de test, tout étant simulable avec mon méta-simulateur, même si des versions avancées ne pourront être effectivement construites pendant 5 ans encore à compter d'aujourd'hui (et alors, nous pourrons demander des avancées supplémentaires juste en faisant avancer les specs du méta-simulateur d'un cran).

Supposons que vous souhaitiez savoir si cinq unités séparées de multiplicateur et/ou trois branchements d'instructions possibles vont augmenter la vitesse d'un programme MMIX. Ou peut-être si le cache des instructions ou des données devrait être agrandi ou rapetissé ou plus associatif. Alors vous avez juste à déclencher le méta-simulateur et à regarder ce qui se passe.

Andrew : Comme je suppose que vous n'utilisez pas d'unités de test avec MMIXAL, pourriez-vous me dire les étapes par lesquelles vous passez pour être sûr que votre code est correct parmi ces variétés de conditions et d'entrées ? Si vous avez une procédure spécifique de travail autour de la vérification, pourriez-vous la décrire ?

Knuth : La plupart des exemples de code en langage-machine dans TAOCP apparaissent dans les Volumes 1-3 ; pour l'instant, nous nous occupons du Volume 4, un tel niveau de détails de bas niveau est largement non nécessaire, et nous pouvons travailler de façon fiable à un haut niveau d'abstraction. Du coup, je n'ai eu besoin d'écrire seulement qu'une douzaine ou à peu près de programmes MMIX pendant que je préparais les parties en ouverture du Volume 4, et il n'y a rien de substantiel dans ces jolis petits programmes jouets. Pour les petites choses comme ça, j'utilise juste des méthodes de vérification informelles, basées sur la théorie que j'ai écrite pour le livre, conjointement avec l'assembleur de MMIXAL et le simulateur MMIX qui sont déjà disponibles sur le net (et décrits en grand détail dans le livre MMIXware).

Ce simulateur inclut des outils de déboggage comme ceux que j'ai trouvés très utiles dans le système d'Ed Satterthwaite pour ALGOL W, mentionné plus tôt. Je me sens toujours en confiance après avoir testé un programme avec ces outils.

Andrew : Malgré sa formulation il y a de nombreuses années, Teχ continue de prospérer, principalement comme fondation de LaTeχ. Puisque Teχ a effectivement été gelé à votre demande, y a-t-il des fonctionnalités que vous voudriez changer ou ajouter, si vous en aviez le temps et la bande passante ? Dans un tel cas, quels seraient les items principaux que vous ajouteriez ou changeriez ?

Knuth : Je crois que des changements à Teχ causeraient plus de mal que

de bien. Les autres personnes qui veulent d'autres fonctionnalités créent leur propre système, et j'ai toujours encouragé le développement plus avant, mis à part le fait que les personnes devaient donner à leurs programmes un autre nom que celui de mon programme. Je veux avoir la responsabilité permanente de Teχ et Metafont, et pour les détails pratiques qui affectent des documents qui s'appuient sur mon travail, comme les dimensions précises des caractères dans les polices des ordinateurs modernes.

ANDREW : Un des aspects peu discuté du développement de logiciel concerne la manière dont on conçoit le travail sur le logiciel dans un domaine complètement nouveau. Vous vous êtes trouvé face à cette question quand vous avez entrepris Teχ : aucun travail n'existait pour vous comme code source, et c'était un domaine dans lequel vous n'étiez pas un expert. Comment avez-vous démarré la conception, et combien de temps cela a-t-il pris avant que vous ne vous sentiez à votre aise et n'entriez dans la partie programmation ?

KNUTH : C'est une autre bonne question ! J'ai discuté de la réponse en grand détail au Chapitre 10 de mon livre *Programmation littéraire*, ainsi qu'aux Chapitres 1 et 2 de mon livre *Typographie digitale*. Je pense que toute personne réellement intéressée par ces sujets apprécierait de lire ces chapitres (voir aussi les Chapitres 24 et 25 de *Typographie digitale* pour compléter les premier et second brouillons de ma conception initiale de Teχ en 1977).

ANDREW : Les livres sur Teχ et le programme lui-même montre clairement la nécessité de limiter l'usage de la mémoire, un problème important pour les systèmes dans ce domaine. Aujourd'hui, la question de l'utilisation de la mémoire dans les programmes a davantage à voir avec les tailles des caches. En tant que personne ayant conçu un processeur de façon logicielle, les questions sensibles au cache et les algorithmes oublieux du cache ont sûrement dû passer sur vos écrans radar. Est-ce que le rôle des caches des processeurs sur la conception des algorithmes est un sujet que vous allez couvrir, même indirectement, dans votre travail à venir ?

KNUTH : J'ai mentionné précédemment que MMIX fournit un lit de test pour de nombreuses sortes de cache. Et c'est une machine implémentée de façon logicielle, de telle manière que nous pouvons réaliser des expériences qui seront répétables même dans une centaine d'années à compter d'aujourd'hui. Certainement que les prochaines éditions des Volumes 1-3 discuteront

du comportement de quelques algorithmes de base par rapport aux différents paramètres de cache.

Dans le Volume 4 jusque là, je compte environ une douzaine de références à la mémoire cache et aux approches ne posant pas de problèmes de cache (sans mentionner une "memo cache," qui est une idée différente mais en lien dans le logiciel).

ANDREW : Quel ensemble d'outils utilisez-vous aujourd'hui pour écrire TAOCP ? Utilisez-vous Teχ ? LaTeχ ? CWEB ? un éditeur de texte ? Et qu'utilisez-vous pour coder ?

KNUTH : Mon style de travail général consiste à tout écrire d'abord au crayon sur un papier, assis à côté d'une grosse poubelle. Après ça, j'utilise Emacs pour entrer le texte dans ma machine, en utilisant les conventions (balises) de Teχ. J'utilise *tex*, *dvips*, et *gv* pour voir les résultats, qui apparaissent sur mon écran presque instantanément de nos jours. Je vérifie mes maths avec Mathematica.

Je programme tous les algorithmes dont je discute (de manière à les comprendre complètement) en utilisant CWEB, qui marche de manière splendide avec le déboggueur GDB. Je réalise les illustrations avec MetaPost (ou, dans de rares cas, sur un Mac avec Adobe Photoshop ou Illustrator). J'ai quelques outils faits maison, comme mon propre vérificateur orthographique pour Teχ et CWEB dans Emacs. J'ai dessiné ma propre font bitmap, qui est utilisée par Emacs, parce que je hais l'apparence de l'apostrophe ASCII et les guillemets gauche sont devenus des symboles indépendants qui ne se ressemblent pas. J'ai des modes spéciaux dans Emacs pour m'aider à classer les dizaines de milliers de papiers et notes dans mes fichiers, et des raccourcis clavier particuliers dans Emacs qui font que l'écriture de livre ressemble un peu au jeu sur un orgue. Je préfère *rxvt* à *xterm* pour l'entrée au terminal. Depuis décembre dernier, j'ai utilisé un système de backup des fichiers appelé *backupfs*, qui satisfait mon besoin d'archiver chaque état quotidien des fichiers d'une manière merveilleuse.

Selon les répertoires courants sur ma machine, j'ai écrit 68 programmes CWEB différents jusqu'à aujourd'hui, cette année. Il y en avait 100 en 2007, 90 en 2006, 100 en 2005, 90 en 2004, etc. De plus, CWEB a une fonctionnalité

très pratique, le mécanisme “change fichier”, avec laquelle je peux rapidement créer de multiples versions et variations sur un thème ; jusqu'à aujourd'hui en 2008, j'ai fait 73 variations de ces 68 thèmes (quelques-unes des variations sont plutôt courtes, seulement quelques octets ; d'autres pèsent 5 kilo-octets ou plus. Quelques-un des programmes CWEB sont assez substantiels, comme le package BDD [2] de 55 pages que j'ai terminé en janvier). Ainsi, vous pouvez voir combien la programmation littéraire est importante dans ma vie.

J'utilise actuellement un système Ubuntu Linux, sur un pc portable, je n'ai pas de connexion internet. Je transfère occasionnellement des fichiers de cette machine à mon Mac que j'utilise pour le réseau, le surf sur la toile et les graphismes, via des lecteurs de mémoire flash ; mais je ne confie mes bijoux de famille qu'à Linux. Incidemment, avec Linux, je préfère beaucoup le clavier que je peux obtenir avec le classique FVWM aux environnements GNOME et KDE que les autres personnes semblent apprécier davantage. A chacun l'environnement qui lui convient.

Andrew : Vous expliquez dans la préface du Fascicule 0 du Volume 4 de TAOCP que le Volume 4 comprendra sûrement trois volumes et peut-être davantage. Il est clair dans ce texte que vous appréciez vraiment d'écrire sur ce sujet. Ceci étant donné, quelle est la confiance que vous accordez à la note, postée à ce sujet sur le site consacré au TAOCP, que le Volume 5 verra le jour environ en 2015 ?

Knuth : Si vous vérifiez sur la Machine Repars-en-arrière [3] les incarnations précédentes de cette page web, vous verrez que le nombre 2015 n'a pas toujours été constant.
Vous corrigerez sûrement que j'ai une machine à écrire à boule qui écrit tout ce matériau, parce que je continue à courir après des faits fascinants, qui peuvent simplement être omis, même si plus de la moitié de mes notes n'iront pas en finale.
Les estimations précises des durées sont impossible, parce que je ne peux pas dire, avant d'avoir avancé profondément sur une section, combien de matériau dans mes fichiers va être vraiment fondamental et combien ne va pas être intéressant pour mon livre ou bien va être trop avancé. Beaucoup de

2. Base de Données
3. WayBack Machine.

la littérature récente consiste en une mise à niveau académique et est d'un intérêt limité pour moi ; les auteurs, de nos jours, introduisent souvent des méthodes ésotériques qui dépassent les techniques plus simples seulement lorsque la taille du problème excède le nombre de protons dans l'univers. De tels algorithmes ne seraient jamais important dans une application réelle du programme. J'ai lu des centaines de tels papiers pour voir s'ils pourraient contenir certaines pépites pour les programmeurs, mais la plupart d'entre eux sont liquidés sans en obtenir grand chose.

Du point de vue de la planification, tout ce que je sais aujourd'hui, c'est que je dois digérer une énorme quantité du matériau que j'ai collecté et mémorisé pendant 45 ans. Je gagne un temps important en travaillant en mode batch : je ne lis pas les papiers en profondeur jusqu'à ce que j'en aie trouvé une douzaine d'autres sur le même sujet durant la même semaine. Quand je suis finalement prêt à lire ce que j'ai collecté à propos d'un sujet, il peut se trouver que je peux zoomer parce que la plupart du matériau est éminemment oubliable pour atteindre mes objectifs. D'un autre côté, je peux découvrir que c'est fondamental et consacrer des semaines d'étude au sujet ; et alors, je mets à jour mon site web et je note que le nombre 2015 est plus près de l'infini.

Andrew : Fin 2006, on vous a diagnostiqué un cancer de la prostate. Comment allez-vous aujourd'hui ?

Knuth : Naturellement, le cancer est un problème sérieux. J'ai des médecins très compétents. En ce moment, je me sens en aussi bonne santé que toujours, modulo le fait que j'ai 70 ans. Les mots viennent aussi librement quand j'écris TAOCP et quand j'écris des programmes littéraires qui précèdent les brouillons de TAOCP. Je me réveille le matin avec des idées qui me plaisent, et quelques-unes de ces idées me plaisent aussi plus tard dans la journée quand je les ai entrées dans mon ordinateur.

D'un autre côté, je me mets volontairement entre les mains de Dieu par rapport au temps qu'il me reste pour être encore capable de faire des choses avant que le cancer ou une maladie cardio-vasculaire ou la sénilité ou quoi que ce soit d'autre ne frappe. Si je devais mourir de façon inattendue demain, je n'aurais aucune raison de me plaindre, parce que ma vie a été incroyablement bénie. Inversement, tant que je suis capable d'écrire à propos de l'informatique, j'essaie de faire de mon mieux pour organiser et exposer au sujet des

dizaines de milliers de papiers techniques que j'ai accumulés et sur lesquels j'ai écrit des notes depuis 1962.

Andrew : Sur votre site web, vous mentionnez que le service des Archives personnelles a récemment filmé une série de videos dans lesquelles vous revenez sur votre vie passée. Dans le morceau 93, "Conseil aux jeunes gens", vous expliquez que les personnes ne devraient jamais faire quelque-chose uniquement parce que c'est à la mode. Comme nous le savons tous trop bien, le développement logiciel est sujet aux modes comme toute autre discipline. Pouvez-vous donner des exemples qui sont actuellement en vogue, mais que les développeurs ne devraient pas adopter simplement parce qu'il sont populaires en ce moment ou parce que c'est la manière dont ils se font actuellement ? Voudriez-vous identifier des exemples importants, en dehors du développement de logiciel ?

Knuth : Hmm. Cette question est presque contradictoire, parce que je suis en train de conseiller aux jeunes gens de s'écouter eux-mêmes plutôt que d'écouter les autres, et je suis un des autres. Presque toute biographie de toute personne que vous pourriez imiter dira qu'il ou elle a fait beaucoup de choses contre la "sagesse conventionnelle" de l'époque.

De plus, j'ai horreur de rabaisser vos questions même si je déteste également offenser les sensibilités des autres personnes dans la mesure où la méthodologie du logiciel a toujours été proche de la religion. Avec l'avertissement qu'il n'y a pas de raison que quiconque se préoccupe des opinions d'un informaticien/mathématicien comme moi à propos du développement logiciel, laissez-moi juste dire que presque tout ce que j'ai entendu associé avec le terme "programmation extrême" semble être exactement le mauvais chemin à ne pas prendre... à une exception près. L'exception est l'idée de travailler en équipe et de lire le code des autres. Cette idée est cruciale, et elle pourrait même effacer à elle seule tous les aspects terribles de la programmation extrême qui m'alarment.

Je dois également confesser un fort biais contre la mode du code réutilisable. Pour moi, le "code rééditable" est bien, bien mieux que n'importe quelle boîte noire ou boîte à outils. Je pourrais continuer là-dessus encore et encore. Si vous êtes totalement convaincu que le code réutilisable est merveilleux, il est peu probable que je sois capable de vous faire changer d'avis, mais vous ne

me convaincrez jamais que le code réutilisable n'est pas principalement une menace.

Voici une question que vous auriez pu souhaiter poser : pourquoi votre nouveau livre s'appelle-t-il Volume 4 Fascicule 0, plutôt que Volume 4 Fascicule 1 ? La réponse est que les programmeurs de logiciels comprendront que je n'étais pas prêt à commencer l'écriture du Volume 4 de TAOCP à son réel point de démarrage, parce que nous savons que l'initialisation d'un programme ne peut pas être écrite tant que le programme n'a pas pris forme. Ainsi, j'ai commencé en 2005 avec le Volume 4 Fascicule 2, après quoi sont venus les Fascicules 3 et 4 (pensez à Star Wars, qui a commencé à l'Episode 4.)

Finalement, j'étais en disposition psychologique pour écrire les parties du début, mais j'ai rapidement réalisé que les sections introductives avaient besoin de contenir davantage de matériau que ne pouvait en contenir un seul fascicule. Alors, me rappelant la consigne de Dijskstra qui dit que le comptage devrait commencer à 0, j'ai décidé de démarrer le Volume 4 avec un Fascicule 0. Vous chercherez le Volume 4 Fascicule 1 plus tard dans l'année.

Andrew Binstock est analyste principal à Pacific Data Works. Il écrit des articles pour SD Times et contribue comme éditeur senior à InfoWorld magazine.