

PAUL GRAHAM

LES HACKERS & LES PEINTRES

Grandes idées sur
l'ère numérique

Traduit de l'anglais par
AURORE LUCAS
et
DORIAN MARIÉ

Note aux lecteurs :

Les chapitres sont tous indépendants les uns des autres, nul besoin de les lire dans l'ordre, n'hésitez pas à en sauter un qui pourrait vous ennuyer. Si vous êtes confronté.es à un terme technique inconnu, je vous renvoie au glossaire ou au Chapitre 10, qui explique une grande partie des concepts qui sous-tendent le logiciel.

Nous avons le regret de vous informer qu'après la lecture du Chapitre 5, le département des relations publiques de Microsoft n'a pas pu nous accorder l'autorisation de reproduire les travaux et articles de Bill Gates. Nous remercions néanmoins la police d'Albuquerque pour le substitut disponible page 95.

paulgraham.com

SOMMAIRE

Préface	4
1. Pourquoi les nerds ne sont-ils pas populaires ? <i>Ils n'ont pas l'esprit dans le jeu.</i>	7
2. Les Hackers et les Peintres <i>Les hackers sont des créateurs, comme les peintres, les architectes et les écrivains.</i>	22
3. Ce que vous ne pouvez pas dire <i>Comment avoir des pensées hérétiques et que faut-il en faire.</i>	39
4. Bonne Mauvaise Attitude <i>Comme les américains, les hackers gagnent en enfreignant les règles.</i>	56
5. L'autre Voie à Suivre <i>Les logiciels basés sur le web offrent les plus grandes opportunités depuis l'arrivée du micro-ordinateur</i>	63
6. Comment Créer de la Valeur <i>Le meilleur moyen de s'enrichir est de créer de la richesse. Et les startups sont le meilleur moyen d'y parvenir</i>	97
7. Faites Attention à l'Écart <i>La "répartition inégale des revenus" pourrait-elle être un problème moins grave qu'on ne le pense ?</i>	121
8. Un Plan Pour le Spam <i>Jusqu'à récemment, la plupart des experts pensaient que le filtrage des spams ne marcherait pas. Cette proposition les a fait changer d'avis.</i>	134
9. Le Goût pour les Créateurs <i>Comment fabrique-t-on de grandes choses ?</i>	143
10. Les Langages de Programmation Expliqués <i>Qu'est-ce qu'un langage de programmation et pourquoi sont-ils un sujet d'actualité ?</i>	160

11. Le Langage de Cent-Ans	169
<i>Comment programmerons-nous dans cent ans ? Pourquoi ne pas commencer maintenant ?</i>	
12. Battre les Moyennes	184
<i>Pour les applications basées sur le web, vous pouvez utiliser le langage que vous voulez. Il en va de même pour vos concurrents.</i>	
13. La Revanche des Nerds	197
<i>Dans le domaine de la technologie, les “meilleures pratiques de l’industrie” sont une recette pour perdre.</i>	
14. Le Langage de Rêve	218
<i>Un bon langage de programmation est un langage qui permet aux hackers de s’en servir à leur guise.</i>	
15. Design et Recherche	235
<i>La recherche doit être originale. Le design doit être bon.</i>	

PRÉFACE

Ce livre est une tentative d'expliquer au grand public ce qui découle du monde des ordinateurs, donc il n'est pas seulement pour les programmeurs. Par exemple, le Chapitre 6 traite de comment devenir riche. Je crois ne pas me tromper si je dis qu'il s'agit d'un sujet qui intéresse tout le monde.

Vous avez peut-être remarqué que, dans les 30 dernières années, une grande partie des gens à être devenus riches sont des programmeurs. Bill Gates, Steve Jobs, Larry Ellison. Pourquoi ? Pourquoi des programmeurs plutôt que des ingénieurs ou des photographes ou des actuaires ? "Comment Créer de la Valeur" l'explique bien.

L'argent des logiciels est un exemple d'une tendance plus générale, et celle-ci est le thème de cet ouvrage. Il s'agit de l'Ere Numérique. C'était censé être l'Ère Spatiale, ou l'Ère Atomique. Mais ce ne sont que des mots inventés par les responsables des relations publiques. Les ordinateurs ont eu bien plus d'effet sur nos vies que les voyages spatiaux ou la technologie nucléaire.

Tout ce qui est autour de nous se base sur les ordinateurs. La machine à écrire ? Remplacé par un ordinateur. Votre téléphone en est en quelque sorte devenu un, comme les appareils photo. Bientôt ce sera le tour de la télévision. Votre voiture possède plus de puissance de traitement qu'un ordinateur central de la taille d'une salle en 1970. Lettres, encyclopédies, journaux et même les commerces locaux sont peu à peu remplacés par internet. Donc, si vous comprenez où nous en sommes et où on va, cela vous aidera à comprendre ce qu'il se passe dans la tête d'un hacker.

Les hackers ? Ce ne sont pas ces gens qui infiltre les ordinateurs ? Pour les non-concernés, c'est la connotation de ce terme. Mais, au sein du monde numérique des ordinateurs, les programmeurs experts se réfèrent à eux-même en tant que hackers. Et étant donné que le but de cet ouvrage est d'expliquer comment les choses sont réellement dans ce monde, j'ai décidé que le risque d'utiliser ce terme et les autres qui suivront.

Les premiers chapitres répondent à des questions auxquelles nous avons probablement tous pensé, réfléchi. Qu'est-ce qui fait le succès d'une startup ? La technologie créera-t-elle un fossé entre ceux qui la comprennent et ceux qui ne la comprennent pas ? Que font les programmeurs ? Pourquoi des enfants qui ne parviennent pas à finir le lycée peuvent finir par figurer parmi les personnes les plus puissantes du monde ? Microsoft va-t-il s'emparer de l'internet ? Que faire des spams ?

Plusieurs chapitres ultérieurs traitent d'un sujet auquel la plupart des personnes extérieures au monde de l'informatique n'ont pas pensé : les langages de programmation. Pourquoi devriez-vous vous intéresser aux langages de programmation?

Parce que si vous voulez comprendre le bidouillage informatique, c'est le fil à suivre. Comme, si vous voulez comprendre la technologie de 1880, les moteurs à vapeur étaient le fil conducteur.

Les programmes informatiques sont juste des textes. Et le langage choisi détermine ce qui peut être dit. Les langages de programmation sont le reflet de la pensée des programmeurs.

Naturellement, cela a un effet important sur le type de pensée que ces derniers ont. Et cela se voit dans les logiciels qu'il écrivent. Orbitz, le site de voyage, a réussi à s'imposer comme le roi du marché dominé par des compétiteurs bien entraînés. Sabre, qui a détenu les réservations en ligne, et Microsoft. Mais par quel miracle Orbitz a-t-il réussi ? Principalement en utilisant un meilleur langage de programmation.

Les programmeurs tendent à se diviser en plusieurs tribus en fonction du langage qu'ils utilisent. Et d'autant plus par le genre de programmes qu'ils écrivent. Il est donc mal vu de dire qu'un langage est mieux qu'un autre. Mais aucun concepteur de langage ne peut se permettre cette fable de courtoisie. Ce que j'ai à dire sur les programmations des langages va sûrement énerver bon nombre de gens, mais je crois réellement qu'il n'y a pas de meilleure manière de comprendre le bidouillage.

Certains peuvent se poser la question "Qu'est-ce que vous ne pouvez pas dire ?" (Chapitre 3) : quel rapport avec les ordinateurs ? Le fait est que les hackers sont obsédés par la liberté d'expression. *Slashdot*, la version hacker du *New York Times* possède toute une section à ce sujet. Je pense que la plupart des lecteurs de *Slashdot* le prennent pour acquis. Mais *Planes and Pilots* n'a pas de section sur la liberté d'expression.

Pourquoi les hackers se soucient-ils tant de la liberté d'expression ? En partie, je pense, que c'est parce que l'innovation est si importante dans les logiciels, et que l'innovation et l'hérésie sont pratiquement la même chose. Les bons hackers développent l'habitude de tout remettre en question. Il faut le faire quand on travaille sur des machines faites de mots aussi complexes qu'une montre mécanique et mille fois plus grandes.

Mais je pense que les marginaux et les iconoclastes sont aussi plus susceptibles de devenir des hackers. Le monde des ordinateurs est comme un Far West intellectuel, où vous pouvez penser ce que vous voulez, du moment qu'on est prêt à se risquer aux conséquences.

Et ce livre, si je l'ai fait de la manière dont je le voulais, est un Western intellectuel. Je ne voudrais pas que vous lisiez ce livre par devoir, en pensant "Eh bien, ces nerds semblent prendre le contrôle du monde. Je suppose que je ferais mieux de comprendre ce qu'ils font, pour ne pas être pris au dépourvu par ce qu'ils préparent ensuite.". Si vous aimez ces idées, ce livre devrait vous paraître intéressant et amusant. Bien que les hackers aient généralement l'air ennuyeux de l'extérieur, les intérieurs de leur tête sont des endroits étonnamment intéressants.

Cambridge, Massachusetts
Avril 2004

Chapitre 1

Pourquoi les nerds ne sont-ils pas populaires ?

Quand on était en première année de lycée, mon ami Rich et moi-même avons créé une sorte de carte en fonction de la popularité des gens. Ce fut une chose facile à faire étant donné que les élèves ne mangent qu'avec les gens avec la même popularité. Nous les avons noté de A à E. Les tables A étaient composées des footballeurs, des cheerleaders et ainsi de suite. Les tables E étaient quant à elles composées de gens atteints de légers cas de trisomie, qu'on appelait en ces temps-là des "attardés".

Nous nous classions à la table D, le plus bas que vous pouviez vous classer si votre physique n'était qu'ordinaire. Nous n'étions pas si candides en nous plaçant à la table "D". Ce serait mentir délibérément que dire l'inverse. Tous les élèves du collège étaient au courant de la popularité des autres, y compris la nôtre.

Je connais bon nombre de personnes qui étaient des nerds à l'école, et toutes ces personnes disent la même chose : il y a une forte corrélation entre le fait d'être intelligent et être un "nerd", et une corrélation encore plus forte entre être populaire et être un "nerd". Être intelligent fait de vous une personne peu populaire.

Pourquoi me direz-vous ? Pour une personne actuellement à l'école, la question semble un peu étrange. Ce simple fait était tellement suffoquant qu'il était difficile de s'imaginer une autre voie. Mais cela se pourrait. Être intelligent ne fait pas de vous un paria à l'école, aussi bien qu'il ne vous portera pas préjudice dans la vraie vie. Aussi loin que je puisse le dire, les problématiques sont les mêmes dans les autres pays. Dans un lycée basique américain, être intelligent vous rend la vie compliquée. Pourquoi ?

La clef de ce mystère est de légèrement reformuler la question. En effet, pourquoi les enfants dits "intelligents" ne se rendent pas populaires ? S'ils sont si intelligents, pourquoi n'arrivent-ils pas à comprendre fonctionne la popularité et à battre le système comme ils le feraient pour des tests standardisés ?

Un argument dit que cela serait impossible du fait que les autres envient leur intelligence et que quoi qu'ils fassent, rien ne les rendra populaires. Si les autres élèves de lycée m'enviaient, ils ont bien occulté cette facette de leur

personnalité. Qui plus est, si être intelligent était une qualité à envier, les filles auraient cassé les rangs.

Dans les écoles où je suis allé, être intelligent n'avait pas tant d'importance. Les élèves ne l'adulaient ou ne le détestaient pas. Si nous mettons une valeur équivalente, ils auraient choisi d'être du côté intelligent plutôt que du côté des gens dits bêtes. D'un autre point de vue, l'intelligence compte beaucoup moins que l'apparence physique, le charisme ou les capacités athlétiques.

Si l'intelligence elle-même n'est pas un facteur de popularité, pourquoi les enfants intelligents sont constamment peu populaires ? Pour moi, la réponse est qu'ils ne veulent pas être populaires.

Si quelqu'un m'avait dit ça à l'époque, je me serais moqué de lui. Ne pas être populaire à l'école fait d'eux des élèves si misérables que certains en viennent au suicide. En me disant ne pas vouloir être populaire, c'est comme dire que quelqu'un qui mourrait de soif dans un désert refuserait un verre d'eau. Bien sûr que je souhaitais être populaire.

Mais dans les faits, je le voulais, mais pas assez. Il y avait quelque chose que je désirais plus : être intelligent. Pas simplement avoir de bonnes notes à l'école ou avoir des pensées qui compteraient pour quelque chose mais plutôt de construire de belles fusées, de bien écrire ou encore de comprendre comment programmer un ordinateur. De façon plus générale, de faire de belles choses.

A l'époque, je n'ai jamais pensé à séparer mes idées et les confronter les unes aux autres. Si j'avais pu, j'aurais vu qu'être intelligent était bien plus important. Si quelqu'un m'avait offert la possibilité d'être l'enfant le plus populaire de l'école, mais au prix d'être doté d'une intelligence moyenne (notez bien mon humour), je ne l'aurais pas saisi.

Bien que ce manque de popularité les fasse souffrir, je ne pense pas que la majorité des nerds le ferait. Pour eux, l'idée d'être dotée d'une intelligence moyenne leur est insupportable, mais la majorité des enfants aurait saisi l'opportunité. Pour la moitié d'entre eux, il s'agit d'une amélioration. Même pour 80% d'entre eux (en supposant que, comme tout le monde semblait le faire à l'époque, l'intelligence est un échelon), qui ne céderait pas 30 points en échange de l'amour et de l'admiration de tout le monde ?

Et ce, pour moi, est l'origine du problème. Les nerds servent deux maîtres. Ils veulent être populaires, certes, mais ils veulent être encore plus intelligents. Et la popularité n'est pas quelque chose que l'on peut faire dans son

temps libre, ni dans l'environnement férocelement compétitif d'un lycée américain.

Alberti, sans doute l'archétype de l'homme de la Renaissance, écrit "qu'aucun art, aussi mineur soit-il, n'exige moins qu'un dévouement total si l'on veut y exceller"[1]. Je me demande si quelqu'un au monde travaille plus dur que les écoliers américains à la popularité. La Marine ou les résidents en neurochirurgie passent pour des fainéants en comparaison. Ils prennent des vacances, certains ont même des hobbies. Un adolescent américain s'efforce d'être populaire à chaque heure de la journée, 365 jours par an.

Je ne veux pas dire qu'ils le font consciemment. Certains d'entre eux sont vraiment des petits Machiavel, mais ce que je veux dire ici, c'est que les adolescents sont toujours en devoir d'être conformistes.

Par exemple, les adolescents accordent une grande attention aux vêtements. Ils ne s'habillent pas consciemment pour être populaires. Ils s'habillent pour être beaux. Mais pour qui ? Pour les autres enfants. L'opinion des autres devient leur définition du bien, pas seulement pour les vêtements, mais pour presque tout ce qu'ils font, jusqu'à leur façon de marcher. Et c'est ainsi que, chaque effort qu'ils font pour faire les choses "correctement" est aussi, consciemment ou non, un effort pour être plus populaire.

Les nerds ne s'en rendent pas compte. Ils ne réalisent pas que devenir populaire demande du travail. En général, les personnes qui ne travaillent pas dans un domaine très exigeant ne réalisent pas à quel point la réussite dépend d'un effort constant (bien que souvent inconscient). Par exemple, la plupart des gens semblent considérer la capacité à dessiner comme une sorte de qualité innée, comme le fait d'être grand. La taille. En réalité, la plupart des personnes qui "savent dessiner" aiment dessiner et ont passé de nombreuses heures à le faire; c'est pour cela qu'ils sont bons.



Club d'échecs du lycée Gateway, 1981. C'est moi en haut à gauche.

De même, la popularité n'est pas seulement quelque chose que l'on est ou que l'on n'est pas, mais quelque chose que l'on fait soi-même.

La principale raison pour laquelle les nerds sont impopulaires est qu'ils ont d'autres d'autres choses à penser. Leur attention est attirée par les livres ou le ou le monde naturel, et non par les modes et les fêtes. Ils sont comme quelqu'un qui essayait de jouer au football tout en tenant un verre d'eau en équilibre sur sa tête. D'autres joueurs, qui peuvent concentrer toute leur attention sur le jeu, les battent sans effort et se demandent pourquoi ils semblent si incapables.

Même si les nerds se souciaient autant que les autres enfants de leur popularité, être populaire leur demanderait plus d'efforts. Les enfants populaires ont appris à être populaires et à vouloir l'être, de la même façon que les nerds ont appris à être intelligents et à vouloir l'être par leurs parents. Alors que les nerds étaient formés pour obtenir les bonnes réponses, les enfants populaires étaient formés pour obtenir les bonnes réponses.

Jusqu'à présent, j'ai peaufiné la relation entre intelligent et nerd, en les utilisant comme interchangeables. De fait, seul le contexte permet de les interchanger. Un nerd est quelqu'un pas assez adapté socialement. Mais le "pas assez" dépend de là où vous êtes. Dans une école américaine typique, les critères de coolitude sont si élevés (ou du moins si spécifiques) qu'il n'est pas nécessaire d'être particulièrement maladroit pour se faire remarquer (ou du moins, si spécifiques). Il n'est pas nécessaire d'être particulièrement maladroit en comparaison.

Peu d'enfants intelligents peuvent se passer de l'attention qu'exige la popularité. À moins qu'ils ne soient beaux, sportifs ou frères et sœurs d'enfants populaires, ils auront tendance à devenir des intellos. C'est la raison pour laquelle pourquoi la vie des gens intelligents est la pire entre, disons, onze et dix-sept ans. À cet âge, la vie tourne beaucoup plus autour de la popularité qu'avant ou après.

Avant cela, la vie des enfants est dominée par leurs parents, et non par les autres enfants. Les enfants se soucient de ce que pensent leurs pairs à l'école élémentaire, mais ça ne relève pas de leur vie entière, ce qui le devient plus tard. Vers l'âge de onze ans, cependant, les enfants semblent commencer à considérer leur famille comme un travail à temps plein. Ils créent un nouveau monde entre eux, et c'est la position dans ce monde qui compte, pas la position dans leur famille. En effet, le fait d'avoir des problèmes dans leur famille peut leur faire gagner des points dans le monde qui leur tient à cœur.

Le problème, c'est que le monde que ces enfants se créent est d'abord très rudimentaire. Si vous laissez une bande de jeunes de onze ans à leurs propres moyens, vous obtenez une version de *Sa Majesté des mouches*. Comme beaucoup d'enfants américains, j'ai lu ce livre à l'école. Ce n'était sans doute pas une coïncidence. On peut supposer que quelqu'un voulait nous faire comprendre que nous étions des sauvages et que nous avions créé un monde cruel et stupide.

C'était trop subtil pour moi. Bien que le livre semblait tout à fait crédible, je n'ai pas compris le message véhiculé par l'auteur. J'aurais préféré qu'il nous dise tout simplement que nous étions des sauvages et que notre monde était stupide.

Les nerds trouveraient leur impopularité plus supportable si elle leur permettait simplement d'être ignorés. Malheureusement être impopulaire à l'école s'accompagne activement de persécutions.

Pourquoi ? Encore une fois, un individu actuellement au lycée pourrait penser qu'il s'agit d'une étrange question. Comment les choses pourraient être autrement ? Elles le pourraient. Les adultes ne persécutent généralement pas les nerds . Mais pourquoi les adolescents oui ?

En partie parce que les adolescents restent globalement des enfants, et certains enfants sont simplement intrinsèquement cruels. Certains torture les nerds comme d'autres arrachent les pattes à une araignée. Avant le développement de la conscience, la torture est amusante.

Une autre raison pour laquelle les enfants persécutent les nerds est qu'ils se sentent mieux. Lorsque vous marchez dans l'eau, vous vous soulevez en poussant l'eau vers le bas. De même, dans toute hiérarchie sociale, les personnes qui ne sont pas sûres de leur position essaieront de la renforcer en maltraitant les autres. J'ai lu que c'est la raison pour laquelle les Blancs pauvres aux États-Unis sont le groupe le plus hostile envers les Noirs.

Mais je pense que la principale raison pour laquelle les autres enfants persécutent les nerds est que cela fait partie du mécanisme de popularité. La popularité n'est que partiellement à propos de l'attrait individuel. C'est bien plus une question d'alliances. Pour devenir plus populaire, vous devez constamment faire des choses qui vous rapprochent d'autres personnes populaires, et rien ne rapproche plus les gens qu'un ennemi commun.

Comme un politicien qui veut distraire les électeurs de la mauvaise conjoncture de son pays, on peut créer un ennemi s'il n'y en a pas de réel. En en persécutant un nerd, un groupe d'enfants plus haut placés dans la hiérarchie créée des liens entre eux. plus haut dans la hiérarchie. L'attaque d'un outsider fait

d'eux tous des insiders. C'est pourquoi les pires cas de persécution se produisent en groupe. Demandez à n'importe quel nerd : un groupe d'enfants vous traite bien plus mal que de n'importe quel intimidateur individuel, aussi sadique soit-il.

Si cela peut consoler les nerds, il n'y a rien de personnel. Le cercle d'enfants qui s'unissent pour s'en prendre à vous fait la même chose, et pour la même raison, qu'une bande de gars qui se réunissent pour aller à la chasse. Ils ne vous détestent pas vraiment. Ils ont juste besoin de quelque chose à chasser.

Parce qu'ils sont au bas de l'échelle, les nerds sont une cible sûre pour toute l'école. Si je me souviens bien, les enfants les plus les plus populaires ne persécutent pas les nerds, ils n'ont pas besoin de s'abaisser à de telles choses. La plupart des persécutions viennent des enfants du bas de l'échelle les classes moyennes nerveuses.

Le problème, c'est qu'il y en a beaucoup. La distribution de la popularité n'est pas une pyramide, mais se rétrécit vers le bas comme une poire. Le groupe le moins populaire est assez petit (je crois que nous étions la seule table D sur la carte de notre cafétéria). Il y a donc plus de gens qui veulent s'en prendre aux nerds qu'il n'y a de nerds.

Tout comme on gagne des points en s'éloignant des enfants impopulaires, on en perd en étant proche d'eux. Une femme que je connais raconte qu'au lycée, elle aimait bien les nerds, mais qu'elle avait peur d'être vue en train de leur parler parce que les autres filles se seraient moquées d'elle. L'impopularité est une maladie contagieuse ; les enfants trop gentils pour s'en prendre aux nerds seront toujours mis à l'écart.

Il n'est donc pas étonnant que les enfants intelligents aient tendance à être malheureux au collège et au lycée. Leurs autres centres d'intérêt ne leur laissent que peu d'attention à consacrer à la popularité, et comme la popularité ressemble à un jeu à somme nulle, cela fait d'eux des cibles pour l'ensemble de l'école. Et ce qui est étrange, c'est que ce scénario cauchemardesque se produit sans aucune malveillance consciente, simplement en raison de la forme de la situation.

Pour moi, la pire période a été le collège, quand la culture des enfants était nouvelle et dure, et que la spécialisation qui séparerait plus tard progressivement les enfants les plus intelligents n'avait pas encore commencé. Presque tous ceux à qui j'ai parlé sont d'accord : le point le plus bas se situe quelque part entre onze et quatorze ans.

Dans notre école, il s'agissait de la quatrième, ce qui correspondait à l'âge de douze et treize ans pour moi. Il y a eu une brève sensation cette année-là

quand l'une de nos enseignantes a surpris un groupe de filles attendant le bus scolaire, et fut tellement choquée que le lendemain, elle consacra toute la classe à un éloquent plaidoyer pour que les filles ne soient pas aussi cruelles les unes envers les autres.

Il n'a pas eu d'effet notable. Ce qui m'a frappé à l'époque c'est qu'elle était surprise. Vous voulez dire qu'elle ne connaît pas le genre de choses qu'ils se disent ? Vous voulez dire que ce n'était pas normal ?

Il est important de comprendre que, non, les adultes ne savent pas ce que les enfants se font les uns aux autres. Ils savent, dans l'abstrait, que les enfants sont monstrueusement cruels les uns envers les autres, tout comme nous savons dans l'abstrait que des gens sont torturés dans les pays pauvres. Mais, comme nous, ils n'aiment pas s'attarder sur ce fait déprimant, et ils ne voient pas de preuves d'abus spécifiques à moins qu'ils ne cherchent à en savoir plus.

Les enseignants des écoles publiques sont dans la même situation que les gardiens de prison. La principale préoccupation des gardiens est de maintenir les prisonniers dans les locaux. Ils doivent également les nourrir et, dans la mesure du possible, les empêcher de s'entretuer. En outre, ils veulent avoir le moins possible à faire avec les prisonniers, et ils les laissent donc créer l'organisation sociale qu'ils souhaitent. D'après ce que j'ai lu, la société créée par les prisonniers est tordue, sauvage et omniprésente, et ce n'est pas drôle d'être au fond de cette société.

Dans les grandes lignes, c'était la même chose dans les écoles que j'ai fréquentées. La chose la plus importante était de rester sur place. Pendant ce temps, les autorités vous nourrissaient, empêchaient toute violence manifeste et s'efforçaient de vous enseigner quelque chose. Mais au-delà de cela, ils ne voulaient pas avoir à trop faire avec les enfants. Comme des gardiens de prison, les enseignants nous laissaient le plus souvent livrés à nous-mêmes. Et, comme les prisonniers, la culture que nous avons créée était barbare.

Pourquoi le monde réel est-il plus accueillant pour les nerds ? On pourrait croire que la réponse est simplement qu'il est peuplé d'adultes, trop matures pour se moquer les uns des autres. Mais je ne pense pas que ce soit vrai. Les adultes en prison s'en prennent certainement les uns aux autres. Il en va de même, apparemment, pour les épouses de la société. Dans certains quartiers de Manhattan, la vie des femmes ressemble à une continuation du lycée, avec toutes les mêmes petites intrigues.

Je pense que ce qui est important dans le monde réel, ce n'est pas qu'il soit peuplé d'adultes, mais qu'il soit très vaste, et que les choses que vous faites

ont de réels effets. C'est ce qui manque à l'école, à la prison et aux femmes apprêtables. Les habitants de ces mondes sont enfermés dans de petites bulles où rien de ce qu'ils font ne peut avoir plus qu'un effet local. Naturellement, ces sociétés dégénèrent en sauvagerie. Elles n'ont pas de fonction à suivre pour leur forme.

Lorsque les choses que vous faites ont des effets réels, il ne suffit plus d'être plaisant. Il commence à être important d'obtenir les bonnes réponses, et c'est là que les nerds se montrent à leur avantage. Bill Gates vient évidemment à l'esprit. Bien qu'il soit notoirement dépourvu de compétences sociales, il obtient les bonnes réponses, du moins, en termes de revenus.

L'autre particularité du monde réel est qu'il est beaucoup plus vaste. Dans un bassin suffisamment vaste, même les plus petites minorités peuvent atteindre une masse critique si elles se regroupent. Dans le monde réel, les nerds se rassemblent dans certains endroits et forment leurs propres sociétés où l'intelligence est la chose la plus importante. Parfois le courant commence à passer même dans l'autre sens : parfois, notamment dans les départements de mathématiques et de sciences des universités, les nerds se rassemblent pour former des sociétés où l'intelligence est la chose la plus importante. John Nash admirait tellement Norbert Wiener qu'il a pris son habitude de toucher le mur en marchant dans un couloir.

À treize ans, je n'avais pas beaucoup plus d'expérience du monde que ce que je voyais immédiatement autour de moi. Le petit monde déformé dans lequel nous vivions était, à mon avis, le monde. Le monde semblait cruel et ennuyeux, et je ne sais pas lequel des deux était le pire.

Parce que je ne m'intégrais pas dans ce monde, je pensais qu'il devait avoir un problème avec moi. Je n'ai pas réalisé que la raison pour laquelle nous les nerds ne s'intégraient pas, c'est que d'une certaine manière, nous avions une longueur d'avance. Nous pensions déjà au genre de choses qui comptent dans le monde réel, au lieu de passer tout notre temps à jouer à un jeu exigeant mais surtout sans intérêt, comme les autres.

Nous étions un peu comme un adulte qui se retrouverait au collège. Il ne saurait pas quels vêtements porter, la bonne musique aimer, le bon argot à utiliser. Il aurait semblé aux yeux des enfants un véritable extraterrestre. Le truc, c'est qu'il en saurait assez pour ne pas se soucier de ce qu'ils pensaient. Nous n'avions pas cette confiance.

Beaucoup de gens semblent penser qu'il est bon pour les enfants intelligents d'être jetés avec des enfants "normaux" à ce stade de leur vie. C'est

possible. Mais dans certains cas, au moins, la raison pour laquelle les nerds ne s'intègrent pas est que tous les autres sont fous. Je me souviens d'avoir assisté à un "pep rally" dans mon lycée, et d'avoir vu les pom-pom girls lancer l'effigie d'un joueur adverse dans l'assistance pour qu'elle soit mise en pièces. J'avais l'impression d'être un explorateur témoin d'un rituel tribal.

Si je pouvais revenir en arrière et donner quelques conseils à mon fils de treize ans, la principale chose que je lui dirais serait de lever la tête et de regarder autour de lui. Je ne l'ai pas vraiment compris à l'époque, mais le monde dans lequel nous vivions était aussi faux qu'un Twinkie. Pas seulement l'école, mais la ville entière. Pourquoi les gens s'installent-ils en banlieue ? Pour avoir des enfants ! Alors, pas étonnant que la ville semble ennuyeuse et stérile. L'endroit entier était une ville artificielle créée explicitement dans le but d'élever des enfants.

Là où j'ai grandi, j'avais l'impression qu'il n'y avait nulle part où aller et rien à faire. Ce n'est pas un hasard. Les banlieues sont délibérément conçues pour exclure le monde extérieur, parce qu'il contient des choses qui pourraient mettre en danger les enfants.

Quant aux écoles, elles n'étaient que des enclos de ce monde factice. Officiellement, leur but est d'enseigner aux enfants. En fait, leur but premier est de garder les enfants enfermés dans un même endroit pendant une grande partie de la journée pour que les adultes fassent ce qu'ils ont à faire. Et je n'y vois pas d'inconvénient : dans une société industrielle spécialisée, il serait désastreux d'avoir des enfants en liberté.

Ce qui me dérange, ce n'est pas que les enfants soient enfermés dans des prisons, mais (A) qu'ils n'en soient pas informés et (B) que les prisons soient gérées principalement par les détenus. Les enfants sont envoyés passer six ans à mémoriser des faits sans signification dans un monde gouverné par une caste de géants qui dirigent la société qui courrent après une balle de football américain, comme s'il s'agissait de la chose la plus naturelle au monde. Et s'ils réchignent devant ce cocktail surréaliste, on les appelle des inadaptés.

La vie dans ce monde tordu est stressante pour les enfants. Et pas seulement pour les nerds. Comme toute guerre, elle est préjudiciable même aux vainqueurs. Les adultes ne peuvent s'empêcher de voir que les adolescents sont tourmentés. Alors, pourquoi ne font-ils rien pour y remédier ? Parce qu'ils mettent cela sur le compte de la puberté. Les adultes se disent que si les enfants sont si malheureux, c'est parce que de nouvelles substances chimiques La raison pour laquelle les enfants sont si malheureux, c'est que de nouveaux produits

chimiques monstrueux, les hormones, circulent maintenant dans leur sang et perturbent tout. Il n'y a rien qui cloche dans le système ; il est juste inévitable que les enfants soient malheureux à cet âge.

Cette idée est tellement répandue que même les enfants y croient, ce qui n'aide probablement pas. Quelqu'un qui pense que ses pieds lui font naturellement mal ne va pas s'arrêter pour envisager la possibilité qu'il porte des chaussures à la mauvaise taille.

Je me méfie de cette théorie selon laquelle les enfants de treize ans sont intrinsèquement dérangés. Si c'était physiologique, cela devrait être universel. Les nomades mongols sont-ils tous nihilistes à treize ans ? J'ai lu beaucoup d'histoires, et je n'ai pas vu une seule référence à ce fait prétendument universel avant le vingtième siècle. Les apprentis adolescents de la Renaissance semblent avoir été joyeux et enthousiastes. Ils se bagarraient et se jouaient des tours, bien sûr (Michel-Ange a eu le nez cassé par une brute), mais ils n'étaient pas fous.

Pour autant que je sache, le concept de l'adolescentaux hormones en folie coexiste avec la banlieue. Je ne pense pas qu'il s'agisse d'une coïncidence. Je pense que les adolescents sont rendus fous par la vie qu'on leur fait mener. Les apprentis adolescents de la Renaissance étaient des chiens de travail. Les adolescents d'aujourd'hui sont des chiens d'appartement névrosés. La folie est la folie des désœuvrés partout. Lorsque j'étais à l'école, le suicide était un sujet récurrent parmi les enfants les plus intelligents. Personne que je connaissais ne l'a fait, mais plusieurs en ont eu l'intention, et certains ont peut-être essayé. La plupart du temps, il s'agissait d'une simple pose. Comme d'autres adolescents, nous aimions ce qui était dramatique, et le suicide semblait très dramatique. Mais c'est en partie parce que nos vies étaient parfois véritablement misérables.

Le harcèlement n'était qu'une partie du problème. Un autre problème, peut-être encore plus grave, est que nous n'avons jamais rien eu de concret à travailler. Les humains aiment travailler ; dans la plupart des pays du monde, votre travail est votre identité. Et tout le travail que nous faisions était inutile, ou en tout cas semblait l'être à l'époque.

Au mieux, il s'agissait d'une pratique pour un travail réel que nous pourrions effectuer si loin que nous ne savions même pas à l'époque à quoi nous nous entraînions. Le plus souvent, il s'agissait simplement d'une série arbitraire de cerceaux à franchir, de mots sans contenu conçus principalement pour la testabilité (Les trois principales causes de la guerre civile étaient.... Test : Énumérer les trois principales causes de la guerre civile).

Et il n'y avait aucun moyen de se retirer. Les adultes avaient convenu entre eux que ce serait la voie pour se diriger vers l'université. La seule façon d'échapper à cette vie vide était de s'y soumettre.

Les adolescents avaient un rôle plus actif dans la société. À l'époque préindustrielle, ils étaient tous des apprentis d'une sorte ou d'une autre, que ce soit dans des magasins, dans des fermes ou même sur des navires de guerre. Ils n'ont pas été laissés pour créer leurs propres sociétés. Ils étaient des membres plus jeunes de sociétés adultes.

Les adolescents semblent avoir plus respecté les adultes à ce moment-là, parce que les adultes étaient les experts visibles des compétences qu'ils essayaient d'apprendre. Maintenant, la plupart des enfants ont peu d'idée de ce que leurs parents font dans leurs bureaux éloignés, et ne voient aucun lien (en fait, il y a précisément peu de choses) entre le travail scolaire et le travail qu'ils feront à l'âge adulte.

Et si les adolescents respectaient davantage les adultes, les adultes seraient également plus utiles pour les adolescents. Après quelques années de formation, un apprenti pourrait être d'une réelle aide. Même le plus récent apprenti pourrait être fait pour porter des messages ou balayer l'atelier.

Maintenant, les adultes n'ont pas d'utilité immédiate pour les adolescents. Ils seraient sur le chemin dans un bureau. Ils les déposent donc à l'école sur le chemin du travail, tout comme ils pourraient déposer le chien dans un chenil s'ils partaient pour le week-end.

Que s'est-il passé ? Nous sommes confrontés à un fait dur ici. La cause de ce problème est la même que la cause de tant de maux actuels : la spécialisation. Au fur et à mesure que les emplois deviennent plus spécialisés, nous devons nous former plus longtemps pour eux. Les enfants de l'époque préindustrielle ont commencé à travailler à environ 14 ans au plus tard ; les enfants des fermes, où la plupart des gens vivaient, ont commencé beaucoup plus tôt. Maintenant, les enfants qui vont à l'université ne commencent pas à travailler à temps plein jusqu'à 21 ou 22 ans. Avec certains diplômes, comme les médecins et les doctorats, vous ne terminerez peut-être pas votre formation avant 30 ans.

Les adolescents sont maintenant inutiles, sauf en tant que main-d'œuvre bon marché dans des industries comme la restauration rapide, qui a évolué pour exploiter précisément ce fait. Dans presque tous les autres types de travail, ils seraient une perte nette. Mais ils sont aussi trop jeunes pour être laissés sans surveillance. Quelqu'un doit veiller sur eux, et le moyen le plus efficace de le

faire est de les rassembler en un seul endroit. Ensuite, quelques adultes peuvent tous les regarder.

Si vous vous arrêtez là, ce que vous décrivez est littéralement une prison, bien qu'à temps partiel. Le problème est que de nombreuses écoles s'arrêtent pratiquement là. Le but déclaré des écoles est d'éduquer les enfants. Mais il n'y a pas de pression extérieure pour bien le faire. Et donc la plupart des écoles font un si mauvais travail d'enseignement que les enfants ne le prennent pas vraiment au sérieux, pas même les enfants intelligents. La plupart du temps, nous étions tous, étudiants et enseignants, juste en train de passer par le mouvement.

Dans ma classe de français au lycée, nous étions censés lire *Les Misérables* d'Hugo. Je ne pense pas que l'un d'entre nous connaisse assez bien le français pour se frayer un chemin à travers cet énorme livre. Comme le reste de la classe, j'ai parcouru les Notes de Cliff. Quand on nous a donné un test sur le livre, j'ai remarqué que les questions semblaient bizarres. Ils étaient pleins de longs mots que notre professeur n'aurait pas utilisés. D'où venaient ces questions ? D'après les Notes de Cliff, il s'est avéré. Le professeur les utilisait aussi. Nous faisions tous semblant.

Il y a certainement d'excellents enseignants des écoles publiques. L'énergie et l'imagination de mon professeur de CM1, M. Mihalko, ont fait de cette année quelque chose dont ses élèves parlent encore, trente ans plus tard. Mais les enseignants comme lui étaient des individus nageant en amont. Ils n'ont pas pu réparer le système.

Dans presque tous les groupes de personnes, vous trouverez une hiérarchie. Lorsque des groupes d'adultes se forment dans le monde réel, c'est généralement dans un but commun, et les dirigeants finissent par être ceux qui sont les meilleurs dans ce domaine. Le problème avec la plupart des écoles, c'est qu'elles n'ont pas de but. Mais il doit y avoir une hiérarchie. Et donc les enfants en font une à partir de rien.

Nous avons une phrase pour décrire ce qui se passe lorsque des classements doivent être créés sans aucun critère significatif. Nous disons que la situation dégénère en un concours de popularité. Et c'est exactement ce qui se passe dans la plupart des écoles américaines. Au lieu de dépendre d'un test réel, son rang dépend principalement de sa capacité à augmenter son rang. C'est comme la cour de Louis XIV. Il n'y a pas d'adversaire externe, donc les enfants deviennent les adversaires les uns des autres.

Lorsqu'il y a un véritable test externe de compétence, il n'est pas douloureux d'être au bas de la hiérarchie. Une recrue d'une équipe de football

n'aime pas l'habileté du vétéran; il espère être comme lui un jour et est heureux d'avoir la chance d'apprendre de lui. Le vétéran peut à son tour ressentir un sentiment de *noblesse oblige*. Et plus important encore, leur statut dépend de la façon dont ils s'en sortent contre leurs adversaires, et non de s'ils peuvent pousser l'autre vers le bas.

Les hiérarchies judiciaires sont une toute autre chose. Ce type de société déprécie quiconque y entre. Il n'y a ni admiration en bas, ni *noblesse oblige* en haut. C'est tuer ou être tué.

C'est le genre de société qui se crée dans les écoles secondaires américaines. Et cela se produit parce que ces écoles n'ont aucun but réel au-delà de garder les enfants au même endroit pendant un certain nombre d'heures chaque jour. Ce dont je ne me suis pas rendu compte à l'époque, et en fait je ne l'ai pas réalisé jusqu'à très récemment, c'est que les horreurs jumelles de la vie scolaire, la cruauté et l'ennui, ont toutes deux la même cause.

La médiocrité des écoles publiques américaines a de pires conséquences que de rendre les enfants malheureux pendant six ans. Il engendre une fausseté qui éloigne activement les enfants des choses qu'ils sont censés apprendre.

Comme beaucoup de nerds, probablement, c'était des années après le lycée avant que je puisse me résoudre à lire tout ce qui nous avait été assigné à ce moment-là. Et j'ai perdu plus que des livres. Je me méfiais des mots comme "caractère" et "intégrité" parce qu'ils avaient été tellement défaits par les adultes. Comme ils ont été utilisés alors, ces mots semblaient tous signifier la même chose : l'obéissance. Les enfants qui ont été loués pour ces qualités avaient tendance à être au mieux des taureaux de prix ternes et au pire des schmoozers faciles. Si c'était ce qu'étaient le caractère et l'intégrité, je ne voulais pas en faire partie.

Le mot que j'ai le plus mal compris était "tact". Comme utilisé par les adultes, cela semblait signifier de garder la bouche fermée. J'ai supposé qu'il était dérivé de la même racine que "tacit" et "taciturne", et que cela signifiait littéralement être silencieux. J'ai juré que je ne serais jamais avec tact ; ils n'alleraient jamais me faire taire. En fait, il est dérivé de la même racine que "tactile", et ce que cela signifie, c'est d'avoir une touche habile. Faire preuve de tact est le contraire de maladroit. Je ne pense pas avoir appris cela avant l'université.

Les nerds ne sont pas les seuls perdants dans la course aux rats de popularité. Les nerds sont impopulaires parce qu'ils sont distraits. Il y a d'autres

enfants qui se retirent délibérément parce qu'ils sont tellement dégoûtés par l'ensemble du processus.

Les adolescents, même les rebelles, n'aiment pas être seuls, donc lorsque les enfants se retirent du système, ils ont tendance à le faire en tant que groupe. Dans les écoles où je suis allé, l'accent de la rébellion était mis sur la consommation de drogues, en particulier la marijuana. Les enfants de cette tribu portaient des t-shirts de concert noirs et étaient appelés "freaks".

Les freaks et les nerds étaient des alliés, et il y avait beaucoup de chevauchement entre eux. Les freaks étaient dans l'ensemble plus intelligents que les autres enfants, bien que ne jamais étudier (ou du moins ne jamais paraître) soit une valeur tribale importante. J'étais plus dans le camp de nerds, mais j'étais ami avec beaucoup de freaks.

Ils ont consommé de la drogue, du moins au début, pour les liens sociaux qu'ils ont créés. C'était quelque chose à faire ensemble, et parce que les drogues étaient illégales, c'était un insigne partagé de rébellion.

Je ne prétends pas que les mauvaises écoles soient la raison pour laquelle les enfants ont des soucis avec la drogue. Après un certain temps, les drogues ont leur propre élan. Il ne fait aucun doute que certains des freaks ont finalement utilisé de la drogue pour échapper à d'autres problèmes - des problèmes à la maison, par exemple. Mais, au moins dans mon école, la raison pour laquelle la plupart des enfants ont commencé à consommer de la drogue était la rébellion. Les enfants de quatorze ans n'ont pas commencé à fumer des joints parce qu'ils avaient entendu dire que cela les aiderait à oublier leurs problèmes. Ils ont commencé parce qu'ils voulaient rejoindre une tribu différente.

La mauvaise gouvernance engendre la rébellion ; ce n'est pas une idée nouvelle. Et pourtant, les autorités agissent toujours pour la plupart comme si les drogues étaient elles-mêmes la cause du problème.

Le vrai problème est le vide de la vie scolaire. Nous ne verrons pas de solutions tant que les adultes ne s'en rendront pas compte. Les adultes qui peuvent s'en rendre compte en premier sont ceux qui étaient eux-mêmes des nerds à l'école. Voulez-vous que vos enfants soient aussi malheureux en quatrième que vous ? Je ne le ferai pas. Eh bien, alors, y a-t-il quelque chose que nous puissions faire pour réparer les choses ? Presque certainement. Il n'y a rien d'inévitable dans le système actuel. Cela s'est passé principalement par défaut [2].

Les adultes, cependant, sont occupés. Se présenter aux pièces de théâtre de l'école est une chose. S'en prendre à la bureaucratie éducative en est une

autre. Peut-être que quelques-uns auront l'énergie d'essayer de changer les choses. Je soupçonne que la partie la plus difficile est de réaliser que vous le pouvez.

Les nerds encore à l'école ne devraient pas retenir leur souffle. Peut-être qu'un jour, une force lourde d'adultes se présentera dans des hélicoptères pour vous secourir, mais ils ne viendront probablement pas ce mois-ci. Toute amélioration immédiate de la vie des nerds devra probablement venir des nerds eux-mêmes.

Le simple fait de comprendre la situation dans laquelle ils sont devrait la rendre moins douloureuse. Les nerds ne sont pas des perdants. Ils jouent juste à un jeu différent, et un jeu beaucoup plus proche de celui joué dans le monde réel. Les adultes le savent. Il est difficile de trouver des adultes qui réussissent maintenant qui ne prétendent pas avoir été des nerds au lycée.

Il est important que les nerds se rendent compte aussi que l'école n'est pas la vie. L'école est une chose étrange et artificielle, à moitié stérile et à moitié sauvage. C'est tout compris, comme la vie, mais ce n'est pas la vraie chose. Ce n'est que temporaire, et si vous regardez, vous pouvez voir au-delà même lorsque vous y êtes encore.

Si la vie semble horrible aux enfants, ce n'est ni parce que les hormones vous transforment tous en monstres (comme vos parents le croient), ni parce que la vie est en fait horrible (comme vous le croyez). C'est parce que les adultes, qui n'ont plus aucune utilité économique pour vous, vous ont abandonné pour passer des années enfermés ensemble sans rien de réel à faire. Il est horrible de vivre dans n'importe quelle société de ce type. Vous n'avez pas besoin de chercher plus loin pour expliquer pourquoi les adolescents sont malheureux.

J'ai dit des choses dures dans cet essai, mais la thèse est vraiment optimiste - que plusieurs problèmes que nous tenons pour acquis ne sont en fait pas insolubles après tout. Les adolescents ne sont pas intrinsèquement des monstres malheureux. Cela devrait être une nouvelle encourageante pour les enfants et les adultes.

Chapitre 2

Les Hackers et les Peintres

Quand j'ai terminé mes études supérieures en informatique, j'y suis allé à une école d'art pour étudier la peinture. Beaucoup de gens semblaient surpris que quelqu'un qui s'intéressait aux ordinateurs s'intéresse aussi à la peinture. Ils semblaient penser que le bidouillage informatique et la peinture étaient des types de travail très différents - que le bidouillage était froid, précis et méthodique, et que la peinture était l'expression frénétique d'une certaine envie primitive.

Ces deux images sont erronées. Le bidouillage informatique et la peinture ont beaucoup en commun. En fait, de tous les différents types de personnes que j'ai connues, les pirates et les peintres sont parmi les plus semblables.

Ce que les hackers et les peintres ont en commun, c'est qu'ils sont tous deux des créateurs. Avec les compositeurs, les architectes et les écrivains, ce que les hackers et les peintres essaient de faire, c'est de faire de bonnes choses. Ils ne font pas de recherche en soi, mais si, en essayant de faire de bonnes choses, ils发现 une nouvelle technique, tant mieux.

Je n'ai jamais aimé le terme « informatique ». La principale raison pour laquelle je n'aime pas qu'il n'y en a pas. L'informatique attrape un sac de zones faiblement liées jetées ensemble par un accident de l'histoire, comme la Yougoslavie. À la fin, vous avez des gens qui sont vraiment mathématiciens, mais appelez ce qu'ils font de l'informatique afin qu'ils puissent obtenir des subventions DARPA. Au milieu, vous avez des gens qui travaillent sur quelque chose comme l'histoire naturelle des ordinateurs - qui étudient le comportement des algorithmes pour acheminer les données à travers les réseaux, par exemple. Et puis à l'autre extrême, vous avez les hackers, qui essaient d'écrire des logiciels intéressants, et pour qui les informatiques ne sont qu'un moyen d'expression, comme le béton est pour les architectes ou de la peinture pour les peintres. C'est comme si les mathématiciens, les physiciens et les architectes devaient tous être dans le même département.

Parfois, ce que font les hackers est appelé « génie logiciel », mais ce terme est tout aussi trompeur. Les bons concepteurs de logiciels ne sont pas plus

des ingénieurs que les architectes. La frontière entre l'architecture et l'ingénierie n'est pas bien définie, mais elle est là. Cela se situe entre quoi et comment : les architectes décident de ce qu'il faut faire, et les ingénieurs trouvent comment le faire.

Quoi et comment ne doivent pas être gardés trop séparés. Vous demandez des ennuis si vous essayez de décider quoi faire sans comprendre comment le faire. Mais le bidouillage informatique peut certainement être plus que simplement décider comment mettre en œuvre certaines spécifications. À son meilleur, il crée la spécification, bien qu'il s'avère que la meilleure façon de le faire est de la mettre en œuvre.

Peut-être qu'un jour, « l'informatique », comme la Yougoslavie, se mettra dans ses composantes. Cela pourrait être une bonne chose. Surtout si cela signifiait l'indépendance pour mon pays natal, le bidouillage.

Regrouper tous ces différents types de travail en un seul partage peut être pratique sur le plan administratif, mais c'est déroutant intellectuellement. C'est l'autre raison pour laquelle je n'aime pas le nom de "l'informatique". On peut soutenir que les gens du milieu font quelque chose comme une science expérimentale. Mais les gens à chaque extrémité, les hackers et les mathématiciens, ne font pas vraiment de science.

Les mathématiciens ne semblaient pas s'en préoccuper. Ils se sont heureusement mis au travail en prouvant des théorèmes comme les autres mathématiciens du département de mathématiques, et ont probablement bientôt cessé de remarquer que le bâtiment dans lequel ils travaillent dit "informatique" à l'extérieur. Mais pour les hackers, cette étiquette est un problème. Si ce qu'ils font s'appelle la science, cela leur donne le sentiment qu'ils devraient agir scientifiquement. Donc, au lieu de faire ce qu'ils veulent vraiment faire, c'est-à-dire concevoir de beaux logiciels, les hackers des universités et des laboratoires de recherche pensent qu'ils devraient écrire des articles de recherche.

Dans le meilleur des cas, les documents ne sont qu'une formalité. Les hackers écrivent des logiciels cool, puis écrivent un article à ce sujet, et le papier devient un proxy pour la réalisation représentée par le logiciel. Mais

souvent, cette inadéquation cause des problèmes. Il est facile de s'éloigner de la construction de belles choses pour construire des choses laides qui rendent les sujets plus appropriés pour les documents de recherche.

Malheureusement, les belles choses ne sont pas toujours les meilleurs sujets de recherche. Premièrement, la recherche doit être originale - et comme tous ceux qui ont écrit une thèse ou un doctorat le savent, la façon d'être sûr d'explorer un territoire vierge est de jalonner un terrain que personne ne veut. Deuxièmement, la recherche doit être substantielle - et les systèmes maladroits produisent des documents plus charnus, parce que vous pouvez écrire sur les obstacles que vous devez surmonter pour faire avancer les choses. Il n'y a rien de tel pour résoudre les problèmes que de commencer avec des hypothèses erronées. La majeure partie de l'IA est un exemple de cette règle ; si vous supposez que la connaissance peut être représentée comme une liste d'expressions logiques de prédicat dont les arguments représentent des concepts abstraits, vous aurez beaucoup d'articles à écrire sur la façon de faire en sorte que cela fonctionne. Comme le disait Ricky Ricardo, « Lucy, tu as beaucoup d'explications à faire. »

La façon de créer quelque chose de beau est souvent de faire des ajustements subtils à quelque chose qui existe déjà, ou de combiner des idées existantes d'une manière légèrement nouvelle. Ce genre de travail est difficile à transmettre dans un document de recherche.

Alors, pourquoi les universités et les laboratoires de recherche continuent-ils de juger les hackers par les publications ? Pour la même raison que "l'aptitude scolaire" est mesurée par des tests standardisés simples d'esprit, ou la productivité des programmeurs par des lignes de code. Ces tests sont faciles à appliquer, et il n'y a rien d'autre tant qu'un test facile qui fonctionne.

Mesurer ce que les hackers essaient réellement de faire, concevoir de beaux logiciels, serait beaucoup plus difficile. Vous avez besoin d'un bon sens du design pour juger du bon design. Et il n'y a pas de corrélation, sauf peut-être négative, entre la capacité des gens à reconnaître un bon design et leur confiance qu'ils peuvent le faire.

Le seul test externe est le temps. Au fil du temps, les belles choses ont tendance à prospérer, et les choses laides ont tendance à être jetées.

Malheureusement, le temps nécessaire peut être plus long que les vies humaines. Samuel Johnson a déclaré qu'il a fallu cent ans pour que la représentation d'un écrivain convergeant [1]. Il faut attendre que les amis influents de l'écrivain meurent, puis que tous leurs disciples meurent.

Je pense que les hackers doivent juste se résigner à avoir une grande composante aléatoire dans leur réputation. En cela, ils ne sont pas différents des autres fabricants. En fait, ils ont de la chance en comparaison. L'influence de la mode n'est pas aussi grande dans le bidouillage que dans la peinture.

Il y a des choses pires que d'avoir des gens qui comprennent mal votre travail. Un danger pire serait que vous vous mépreniez vous-même sur votre travail. Les domaines connexes sont l'endroit où vous allez à la recherche d'idées. Si vous vous trouvez dans le département d'informatique, il y a une tentation naturelle de croire, par exemple, que le bidouillage informatique est la version appliquée de ce que l'informatique théorique est la théorie. Tout le temps que j'étais aux études supérieures, j'avais un sentiment inconfortable au fond de mon esprit que je devrais connaître plus de théorie, et que c'était très négligent de ma part d'avoir oublié toutes ces choses dans les trois semaines suivant l'examen final.

Maintenant, je me rends compte que je me suis trompé. Les hackers doivent comprendre la théorie du calcul à peu près autant que les peintres ont besoin de comprendre la chimie de la peinture. Vous devez savoir comment calculer la complexité du temps et de l'espace, et peut-être aussi le concept d'une machine d'état, au cas où vous voudriez écrire un analyseur. Les peintres doivent se souvenir beaucoup plus de la chimie de la peinture que cela.

J'ai constaté que les meilleures sources d'idées ne sont pas les autres domaines qui ont le mot « ordinateur » dans leur nom, mais les autres domaines habités par des fabricants. La peinture a été une source d'idées beaucoup plus riche que la théorie du calcul.

Par exemple, on m'a appris à l'université qu'il faudrait écrire un programme complètement sur papier avant même de s'approcher d'un

ordinateur. J'ai constaté que je n'avais pas programmé de cette façon. J'ai trouvé que j'aimais programmer assis devant un ordinateur, pas un morceau de papier. Pire encore, au lieu d'écrire patiemment un programme complet et de m'assurer qu'il était correct, j'avais tendance à cracher du code qui était désespérément cassé et à le mettre progressivement en forme. Le débogage, de ce que l'on m'a appris, était une sorte de passe finale où vous avez attrapé des fautes de frappe et des oubliés. La façon dont je travaillais, il semblait que la programmation consistait à déboguer.

Pendant longtemps, je me suis senti mal à ce sujet, tout comme je me suis senti mal de ne pas avoir tenu mon crayon comme ils me l'ont appris à l'école primaire. Si seulement j'avais regardé les autres fabricants, les peintres ou les architectes, j'aurais réalisé qu'il y avait un nom pour ce que je faisais : faire des croquis. Pour autant que je sache, la façon dont ils m'ont appris à programmer à l'université était tout à fait fausse. Vous devriez comprendre les programmes au fur et à mesure que vous les écrivez, tout comme les écrivains, les peintres et les architectes.

Réaliser cela a de réelles implications pour la conception de logiciels. Cela signifie qu'un langage de programmation devrait, avant tout, être malléable. Un langage de programmation est pour penser à des programmes, pas pour exprimer des programmes auxquels vous avez déjà pensé. Ce devrait être un crayon, pas un stylo. La dactylographie statique serait une bonne idée si les gens écrivaient réellement des programmes comme ils me l'ont appris à l'université. Mais ce n'est pas comme ça que les hackers que je connais écrivent des programmes. Nous avons besoin d'un langage qui nous permet de griffonner et de frotter et de frotter, pas d'un langage où vous devez vous asseoir avec une tasse de thé de types équilibrés sur votre genou et avoir une conversation polie avec une vieille tante stricte d'un compilateur.

Alors que nous sommes sur le sujet de la dactylographie statique, l'identification avec les fabricants nous sauvera d'un autre problème qui afflige les sciences : l'envie des mathématiques. Tout le monde dans les sciences croit secrètement que les mathématiciens sont plus intelligents qu'ils ne le sont. Je pense que les mathématiciens le croient aussi. Quoi qu'il en soit, le résultat est que les scientifiques ont tendance à rendre leur travail aussi mathématique que possible. Dans un domaine comme la physique, cela ne fait probablement pas

beaucoup de mal, mais plus vous vous en tirez des sciences naturelles, plus cela devient un problème.

Une page de formules a l'air si impressionnante. (Astuce : pour plus d'impression, utilisez des variables grecques). Et il y a donc une grande tentation pour travailler sur des problèmes que vous pouvez traiter formellement, plutôt que sur des problèmes qui sont, disons, importants.

Si les hackers s'identifiaient à d'autres fabricants, comme les écrivains et les peintres, ils ne se sentirait pas tentés de le faire. Les écrivains et les peintres ne souffrent pas de l'envie des mathématiques. Ils ont l'impression de faire quelque chose qui n'a aucun rapport. Il en est de plus que les hackers, je pense.

Si les universités et les laboratoires de recherche empêchent les hackers de faire le genre de travail qu'ils veulent faire, peut-être que la place pour eux est dans les entreprises. Malheureusement, la plupart des entreprises ne laisseront pas non plus les hackers faire ce qu'ils veulent. Les universités et les laboratoires de recherche les forcent à être des scientifiques, et les entreprises les forcent à être des ingénieurs.

Je ne l'ai découvert moi-même que très récemment. Quand Yahoo a acheté Viaweb, ils m'ont demandé ce que je voulais faire. Je n'avais jamais beaucoup aimé les affaires, et j'ai dit que je voulais juste bidouiller. Quand je suis arrivé à Yahoo, j'ai découvert que ce que le bidouillage informatique signifiait pour eux, c'était d'implémenter un logiciel, et non de le concevoir. Les programmeurs étaient considérés comme des techniciens qui traduisaient les visions (si c'est le mot) des chefs de produit en code.

Cela semble être le plan par défaut dans les grandes entreprises. Ils le font parce que cela diminue l'écart-type du résultat. Seul un faible pourcentage de hackers peut réellement concevoir des logiciels, et il est difficile pour les personnes qui dirigent une entreprise de les choisir. Ainsi, au lieu de confier l'avenir du logiciel à un brillant hacker, la plupart des entreprises mettent les choses en place de manière à ce qu'il soit conçu par un comité, et les hackers ne font que mettre en œuvre la conception.

Si vous voulez gagner de l'argent à un moment donné, souvenez-vous de cela, parce que c'est l'une des raisons pour lesquelles les startups gagnent. Les grandes entreprises veulent diminuer l'écart-type des résultats de conception parce qu'elles veulent éviter les catastrophes. Mais lorsque vous amortissez les oscillations, vous perdez les points hauts ainsi que les points bas. Ce n'est pas un problème pour les grandes entreprises, car elles ne gagnent pas en fabriquant d'excellents produits. Les grandes entreprises gagnent en aspirant moins que les autres grandes entreprises.

Donc, si vous pouvez trouver un moyen d'entrer dans une guerre de design avec une entreprise assez grande pour que son logiciel soit conçu par des gestionnaires de produits, ils ne seront jamais en mesure de vous suivre. Cependant, ces opportunités ne sont pas faciles à trouver. Il est difficile d'engager une grande entreprise dans une guerre de design, tout comme il est difficile d'engager un adversaire à l'intérieur d'un château dans un combat au corps à corps. Il serait assez facile d'écrire un meilleur traitement de texte que Microsoft Word, par exemple, mais Microsoft, dans le château de son monopole du système d'exploitation, ne le remarquerait probablement même pas si vous le faisiez.

L'endroit pour mener des guerres de design se trouve sur de nouveaux marchés, où personne n'a encore réussi à établir des fortifications. C'est là que vous pouvez gagner gros en adoptant l'approche audacieuse de la conception et en ayant les mêmes personnes qui conçoivent et mettent en œuvre le produit. Microsoft eux-mêmes l'ont fait au début. Apple aussi. Et Hewlett-Packard. Je soupçonne que presque toutes les start-ups à succès l'ont fait.

Donc, une façon de créer d'excellents logiciels est de démarrer votre propre start-up. Il y a cependant deux problèmes avec cela. La première est que dans une start-up, vous devez faire tellement de choses en plus d'écrire des logiciels. Chez Viaweb, j'ai eu de la chance si je devais bidouiller un quart du temps. Et les choses que j'ai dû faire les trois autres quarts du temps allaient de fastidieuses à terrifiantes. J'ai une référence pour cela, parce que j'ai déjà dû quitter une réunion du conseil d'administration pour faire remplir quelques cavités. Je me souviens m'être assis sur la chaise du dentiste, en attendant la fraise et d'avoir l'impression d'être en vacances.

L'autre problème avec les startups est qu'il n'y a pas beaucoup de tour entre le type de logiciel qui rapporte de l'argent et le type qui est intéressant à écrire. Les langages de programmation sont intéressants à écrire, et le premier produit de Microsoft en était un, en fait, mais personne ne paiera pour les langages de programmation maintenant. Si vous voulez gagner de l'argent, vous avez tendance à être obligé de travailler sur des problèmes qui sont trop désagréables pour que quelqu'un puisse les résoudre gratuitement.

Tous les fabricants sont confrontés à ce problème. Les prix sont déterminés par l'offre et la demande, et il n'y a tout simplement pas autant de demande pour des choses sur lesquelles il est amusant de travailler que pour des choses qui résolvent les problèmes banaux des clients individuels. Agir dans des pièces de théâtre off-Broadway ne paie pas aussi bien que de porter un costume de gorille dans le stand de quelqu'un lors d'un salon professionnel. Écrire des romans ne paie pas aussi bien que d'écrire des textes publicitaires pour l'élimination des ordures. Et le bidouillage informatique des langages de programmation ne paie pas aussi bien que de trouver comment connecter l'ancienne base de données d'une entreprise à son serveur web.

Je pense que la réponse à ce problème, dans le cas des logiciels, est un concept connu de presque tous les fabricants : le travail de jour. Cette phrase a commencé avec des musiciens, qui se produisent la nuit. Plus généralement, cela signifie que vous avez un type de travail que vous faites pour l'argent, et un autre pour l'amour.

Presque tous les fabricants ont un emploi de jour au début de leur carrière. Les peintres et les écrivains le font notamment. Si vous avez de la chance, vous pouvez obtenir un emploi de jour étroitement lié à votre vrai travail. Les musiciens semblent souvent travailler dans les magasins de disques. Un hacker travaillant sur un langage de programmation ou un système d'exploitation pourrait également être en mesure d'obtenir un emploi de jour en l'utilisant [2].

Quand je dis que la réponse est que les hackers ont des emplois de jour et travaillent sur de beaux logiciels sur le côté, je ne propose pas cela comme une nouvelle idée. C'est ce qu'est le piratage open source. Ce que je dis, c'est que

l'open source est probablement le bon modèle, car il a été confirmé de manière indépendante par tous les autres fabricants.

Il me semble surprenant que n'importe quel employeur soit réticent à laisser les hackers travailler sur des projets open source. Chez Viaweb, nous aurions été réticents à embaucher quelqu'un qui ne l'avait pas fait. Lorsque nous avons interviewé les programmeurs, la principale chose dont nous nous soucions était le type de logiciel qu'ils écrivaient pendant leur temps libre. Vous ne pouvez rien faire vraiment bien à moins que vous ne l'aimiez, et si vous aimez bidouiller, vous travaillerez inévitablement sur vos propres projets [3].

Parce que les hackers sont des créateurs plutôt que des scientifiques, le bon endroit pour chercher des métaphores n'est pas dans les sciences, mais parmi d'autres types de fabricants. Qu'est-ce que la peinture peut nous apprendre d'autre sur le bidouillage informatique ?

Une chose que nous pouvons apprendre, ou du moins confirmer, de l'exemple de la peinture est comment apprendre à bidouiller. Vous apprenez à peindre principalement en le faisant. Idem pour le bidouillage. La plupart des hackers n'apprennent pas à bidouiller en suivant des cours universitaires de programmation. Ils apprennent en écrivant leurs propres programmes à l'âge de treize ans. Même dans les cours universitaires, vous apprenez à bidouiller principalement en bidouillant [4].

Parce que les peintres laissent derrière eux une traînée de travail, vous pouvez les regarder apprendre en faisant. Si vous regardez l'œuvre d'un peintre dans l'ordre chronologique, vous constaterez que chaque peinture s'appuie sur des choses apprises dans les précédentes. Lorsqu'il y a quelque chose dans une peinture qui fonctionne particulièrement bien, vous pouvez généralement en trouver la version 1 sous une forme plus petite dans une peinture antérieure.

Je pense que la plupart des fabricants travaillent de cette façon. Les écrivains et les architectes le sont aussi. Peut-être serait-il bon pour les hackers d'agir plus comme des peintres, et de recommencer régulièrement à zéro, au lieu de continuer à travailler pendant des années sur un projet, et d'essayer d'incorporer toutes leurs idées ultérieures en tant que révisions.

Le fait que les hackers apprennent à bidouiller en le faisant est un autre signe de la différence entre le piratage des sciences. Les scientifiques n'apprennent pas la science en le faisant, mais en faisant des laboratoires et des ensembles de problèmes. Les scientifiques commencent à faire un travail parfait, dans le sens où ils essaient juste de reproduire le travail que quelqu'un d'autre a déjà fait pour eux. Finalement, ils en arrivent au point où ils peuvent faire un travail original. Alors que les hackers, dès le début, font un travail original ; c'est juste très mauvais. Ainsi, les hackers commencent l'original, et obtiennent de bons, et les scientifiques commencent bien, et obtiennent l'original.

L'autre façon dont les fabricants apprennent est à partir d'exemples. Pour un peintre, un musée est une bibliothèque de référence de techniques. Pendant des centaines d'années, il fait partie de l'éducation traditionnelle des peintres de copier les œuvres des grands maîtres, car la copie vous oblige à regarder de près la façon dont une peinture est faite.

Les écrivains le font aussi. Benjamin Franklin a appris à écrire en résumant les points des essais d'Addison et de Steele, et en puis en essayant de les reproduire. Raymond Chandler a fait la même chose avec les romans policiers.

De même, les hackers peuvent apprendre à programmer en regardant de bons programmes, non seulement sur ce qu'ils font, mais aussi sur le code source. L'un des avantages les moins médiatisés du mouvement open source est qu'il a facilité l'apprentissage de la programmation. Lorsque j'ai appris à programmer, nous avons dû nous fier principalement à des exemples dans les livres. Le seul gros morceau de code disponible à l'époque était Unix, mais même celui-ci n'était pas open source. La plupart des personnes qui ont lu la source l'ont lu dans des photocopies illicites du livre de John Lions, qui, bien qu'il ait été écrit en 1977, n'a pas été autorisé à être publié avant 1996.

Un autre exemple que nous pouvons prendre de la peinture est la façon dont les peintures sont créées par un raffinement progressif. Les peintures commencent généralement par un croquis. Peu à peu, les détails sont remplis. Mais il ne s'agit pas simplement d'un processus de remplissage. Parfois, les plans originaux sont erronés. D'innombrables tableaux, lorsqu'on les regarde

aux rayons X, il s'avère que des membres ont été déplacés ou des traits du visage qui ont été réajustés.

Voici un cas où nous pouvons apprendre de la peinture. Je pense que le bidouillage informatique devrait aussi fonctionner de cette façon. Il est irréaliste de s'attendre à ce que les spécifications d'un programme soient parfaites. Vous feriez mieux d'admettre cela à l'avance et d'écrire des programmes d'une manière qui permet aux spécifications de changer à la volée.

(La structure des grandes entreprises rend cela difficile à faire pour elles, alors voici un autre endroit où les start-ups ont un avantage.)

Tout le monde connaît probablement maintenant le danger de l'optimisation prématuée. Je pense que nous devrions être tout aussi inquiets au sujet de la conception prématuée - en décidant trop tôt ce qu'un programme devrait faire.

Les bons outils peuvent nous aider à éviter ce danger. Un bon langage de programmation devrait, comme la peinture à l'huile, faciliter le changement d'avis. La dactylographie dynamique est une victoire ici parce que vous n'avez pas à vous engager dans des représentations de données spécifiques dès le départ. Mais la clé de la flexibilité, je pense, est de rendre le langage très abstrait. Le programme le plus facile à changer est un programme court.



Ginevra de' Benci de Léonard de Vinci, 1474.

Cela ressemble à un paradoxe, mais une grande peinture doit être meilleure qu'elle ne doit l'être. Par exemple, lorsque Léonard de Vinci a peint le

portrait de Ginevra de' Benci à la National Gallery, il a mis un buisson de genièvre derrière sa tête. Il y peignait soigneusement chaque feuille. Beaucoup de peintres auraient pu penser que c'est juste quelque chose à mettre en arrière-plan pour encadrer sa tête. Personne ne le regardera de si près.

Pas Léonard de Vinci. La façon dont il a travaillé sur une partie d'une peinture ne dépendait pas du tout de la façon dont il s'attendait à ce que quelqu'un la regarde. Il était comme Michael Jordan. Implacable.

L'acharnement l'emporte parce que, dans l'ensemble, les détails invisibles deviennent visibles. Quand les gens marchent près du portrait de *Ginevra de' Benci*, leur attention est souvent immédiatement attirée par cela, même s'ils regardent l'étiquette et remarquent qu'elle dit Léonard de Vinci. Tous ces détails invisibles se combinent pour produire quelque chose qui est tout simplement étonnant, comme mille voix à peine audibles qui chantent toutes en harmonie.

Un excellent logiciel, de même, nécessite un dévouement fanatique à la beauté. Si vous regardez à l'intérieur de bons logiciels, vous trouvez que les pièces que personne n'est jamais censé voir sont aussi belles. Quand il s'agit de code, je me comporte d'une manière qui me rendrait admissible aux médicaments sur ordonnance si j'abordais la vie quotidienne de la même manière. Cela me rend fou de voir du code qui est mal en retrait, ou qui utilise des noms de variables laids.

Si un hacker était un simple implémentateur, transformant une spécification en code, alors il pourrait simplement se frayer un chemin d'un bout à l'autre comme quelqu'un qui creuse un fossé. Mais si le hacker est un créateur, nous devons tenir compte de l'inspiration.

Dans le bidouillage informatique, comme la peinture, le travail se fait par cycles. Parfois, vous êtes enthousiasmé par un nouveau projet et vous voulez y travailler seize heures par jour. D'autres fois, rien ne semble intéressant.

Pour faire du bon travail, vous devez tenir compte de ces cycles, car ils sont affectés par la façon dont vous y réagissez. Lorsque vous conduisez une voiture avec une transmission manuelle sur une colline, vous devez parfois reculer de l'embrayage pour éviter de décrocher. Le retour en arrière peut

également empêcher l'ambition de s'installer. Dans la peinture et le bidouillage informatique, il y a certaines tâches qui sont terriblement ambitieuses, et d'autres qui sont d'une routine réconfortante. C'est une bonne idée de sauvegarder des tâches faciles pour les moments où vous seriez autrement bloqué.

Dans le bidouillage informatique, cela peut littéralement signifier sauver des bogues. J'aime le debugging : c'est la seule fois où le bidouillage est aussi simple que les gens le pensent. Vous avez un problème totalement limité, et tout ce que vous avez à faire est de le résoudre. Votre programme est censé faire x. Au lieu de cela, il le fait. Où cela va-t-il mal ? Tu sais que tu vas gagner à la fin. C'est aussi relaxant que de peindre un mur.

L'exemple de la peinture peut nous apprendre non seulement comment gérer notre propre travail, mais aussi comment travailler ensemble. Une grande partie du grand art du passé est le travail de plusieurs mains, bien qu'il n'y ait peut-être qu'un seul nom sur le mur à côté dans le musée. Léonard de Vinci était apprenti dans l'atelier de Verrocchio et a peint l'un des anges de son *Baptême du Christ*. Ce genre de choses était la règle, pas l'exception. Michel-Ange a été considéré comme particulièrement dévoué pour avoir insisté pour peindre toutes les figures sur le plafond de la chapelle Sixtine lui-même.

Pour autant que je sache, lorsque les peintres travaillaient ensemble sur une peinture, ils ne travaillaient jamais sur les mêmes parties. Il était courant pour le maître de peindre les figures principales et pour les assistants de peindre les autres et l'arrière-plan. Mais vous n'avez jamais eu un gars qui peignait sur le travail d'un autre.

Je pense que c'est aussi le bon modèle pour la collaboration dans les logiciels. Ne le poussez pas trop loin. Lorsqu'un morceau de code est piraté par trois ou quatre personnes différentes, dont personne ne le possède vraiment, il finira par être comme une salle commune. Il aura tendance à se sentir sombre et abandonné, et à accumuler du cruft. La bonne façon de collaborer, je pense, est de diviser les projets en modules bien définis, chacun avec un propriétaire défini, et avec des interfaces entre eux qui sont aussi soigneusement conçues et, si possible, aussi articulées que les langages de programmation.

Comme la peinture, la plupart des logiciels sont destinés à un public humain. Et donc les hackers, comme les peintres, doivent faire preuve d'empathie pour faire un très bon travail. Vous devez être en mesure de voir les choses du point de vue de l'utilisateur.

Quand j'étais enfant, on me disait constamment de regarder les choses du point de vue de quelqu'un d'autre. Ce que cela signifiait toujours en pratique, c'était de faire ce que quelqu'un d'autre voulait, au lieu de ce que je voulais. Cela a bien sûr donné une mauvaise réputation à l'empathie, et je me suis fait un point d'honneur de ne pas la cultiver.

Oh que j'avais tort. Il s'avère que regarder les choses du point de vue des autres est pratiquement le secret du succès.

L'empathie ne signifie pas nécessairement être dévoué.e. Loin de ça. Comprendre comment quelqu'un d'autre voit les choses n'implique pas que vous agirez dans son intérêt ; dans certaines situations - en guerre, par exemple - vous voulez faire exactement le contraire [5].

La plupart des fabricants font des choses pour un public humain. Et pour évaluer un public, vous devez comprendre ce dont il a besoin. Presque toutes les plus grandes peintures sont des peintures de personnes, par exemple, parce que les gens sont ce qui intéresse les gens.

L'empathie est probablement la différence la plus importante entre un bon et un grand hacker. Certains sont assez intelligents, mais pratiquement solipsistes en matière d'empathie. Il est difficile pour de telles personnes de concevoir d'excellents logiciels, car elles ne peuvent pas voir les choses du point de vue de l'utilisateur [6].

Une façon de dire à quel point les gens sont bons en matière d'empathie est de les regarder expliquer une question technique à quelqu'un qui n'a pas de formation technique. Nous connaissons probablement tous des gens qui, bien que par ailleurs intelligents, sont juste comiquement mauvais dans ce domaine. Si quelqu'un leur demande lors d'un dîner ce qu'est un langage de programmation, il dira quelque chose comme "Oh, un langage de haut niveau est ce que le compilateur utilise comme entrée pour générer du code objet".

Langue de haut niveau ? Compilateur ? Code objet ? Quelqu'un qui ne sait pas ce qu'est un langage de programmation ne sait évidemment pas non plus ce que sont ces choses.

Une partie de ce que les logiciels ont à faire est de s'expliquer. Donc, pour écrire un bon logiciel, vous devez comprendre à quel point les utilisateurs comprennent peu. Ils vont marcher jusqu'au logiciel sans préparation, et il ferait mieux de faire ce qu'ils pensent qu'il fera, parce qu'ils ne vont pas lire le manuel. Le meilleur système que j'ai jamais vu à cet égard était le Macintosh original, en 1984. Il a fait ce que le logiciel ne fait presque jamais : il a juste fonctionné [7].

Le code source devrait également s'expliquer de lui-même. Si je pouvais amener les gens à se souvenir d'une seule citation sur la programmation, ce serait celle du début de la structure et de l'interprétation des programmes informatiques [8].

Les programmes doivent être écrits pour que les gens les lisent, et seulement accessoirement pour que les machines les exécutent.



Federico da Montefeltro de Piero della Francesca, 1465-66 (détail).

Vous devez avoir de l'empathie non seulement pour vos utilisateurs, mais aussi pour vos lecteurs. C'est dans votre intérêt, parce que vous serez l'un d'entre eux. Beaucoup de hackers ont écrit un programme pour découvrir, à leur retour six mois plus tard, qu'ils n'avaient aucune idée de son fonctionnement. Je connais plusieurs personnes qui ont juré Perl après de telles expériences [9].

Le manque d'empathie est associé à l'intelligence, au point qu'il y a même quelque chose de mode pour cela à certains endroits. Mais je ne pense pas qu'il y ait de corrélation. Vous pouvez bien réussir en mathématiques et en sciences naturelles sans avoir à apprendre l'empathie, et les gens dans ces domaines ont tendance à être intelligents, de sorte que les deux qualités en sont ainsi associées. Mais il y a beaucoup de gens stupides qui sont aussi mauvais en empathie.

Donc, si le bidouillage informatique fonctionne comme la peinture et l'écriture, est-ce aussi cool ? Après tout, vous n'avez qu'une seule vie. Vous pourriez aussi bien le consacrer à travailler sur quelque chose de génial.

Malheureusement, il est difficile de répondre à la question. Il y a toujours un grand décalage dans le prestige. C'est comme la lumière d'une étoile lointaine. La peinture a du prestige maintenant grâce à l'excellent travail que les gens ont fait il y a cinq cents ans. À l'époque, personne ne pensait que ces peintures étaient aussi importantes que nous le faisons aujourd'hui. Il aurait semblé très étrange aux gens en 1465 que Federico da Montefeltro, le duc d'Urbino, soit un jour surtout connu comme le gars au nez étrange dans un tableau de Piero della Francesca.

Donc, bien que j'admette que le bidouillage informatique ne semble pas aussi cool que la peinture maintenant, nous devons nous rappeler que la peinture elle-même ne semblait pas aussi cool dans ses jours de gloire qu'elle le fait maintenant.

Ce que nous pouvons dire avec une certaine confiance, c'est que ce sont les jours de gloire du piratage. Dans la plupart des domaines, le grand travail est fait tôt. Les peintures réalisées entre 1430 et 1500 sont inégalées. Shakespeare est apparu juste au moment où le théâtre professionnel était en train de naître, et a poussé le médium si loin que chaque dramaturge a dû vivre dans son ombre. Albrecht Dürer a fait la même chose avec la gravure, et Jane Austen avec le roman.

Encore et encore, nous voyons le même schéma. Un nouveau médium apparaît, et les gens sont tellement enthousiastes qu'ils explorent la plupart de ses possibilités au cours des deux premières générations. Le bidouillage informatique semble être dans cette phase maintenant.

La peinture n'était pas, à l'époque de Léonard de Vinci, aussi cool que son travail l'a fait. La façon dont le piratage s'avère cool dépendra de ce que nous pouvons faire avec ce nouveau média.

Chapitre 3

Ce que vous ne pouvez pas dire

Avez-vous déjà vu une vieille photo de vous-même et été gêné par la façon dont vous aviez l'air ? *Est-ce qu'on s'est vraiment habillé comme ça ?* Nous l'avons fait. Et nous n'avions aucune idée à quel point nous avions l'air idiots. C'est la nature de la mode d'être invisible, de la même manière que le mouvement de la terre est invisible pour nous tous qui montait dessus.

Ce qui me fait peur, c'est qu'il y a aussi des modes morales. Ils sont tout aussi arbitraires et tout aussi invisibles pour la plupart des gens. Mais ils sont beaucoup plus dangereux. La mode est confondue avec le bon design ; la mode morale est confondue avec le bien. S'habiller bizarrement vous fait rire. En violant les modes moraux, vous pouvez être licencié, ostracisé, emprisonné ou même tué.

Si vous pouviez voyager dans une machine à remonter le temps, une chose serait vraie, peu importe où vous alliez : vous devriez regarder ce que vous avez dit. Les opinions que nous considérons comme inoffensives auraient pu vous causer de gros ennuis. J'ai déjà dit au moins une chose qui m'aurait causé de gros ennuis dans la majeure partie de l'Europe au XVII^e siècle, et qui a causé de gros ennuis à Galilée lorsqu'il l'a dit : la terre bouge [1].

Les nerds ont toujours des ennuis. Ils disent des choses inappropriées pour la même raison qu'ils s'habillent de manière démodée et ont de bonnes idées. La Convention à moins de contrôle sur eux.

Cela semble être une constante tout au long de l'histoire : à chaque période, les gens croyaient des choses qui étaient tout simplement ridicules, et les croyaient si fortement que vous auriez eu de terribles ennuis pour avoir dit le contraire.

Est-ce que notre heure est différente ? Pour tous ceux qui ont lu une partie de l'histoire, la réponse est presque certainement non. Ce serait une remarquable coïncidence si la nôtre était la première ère à tout faire correctement.

C'est alléchant de penser que nous croyons des choses que les gens trouveront ridicules à l'avenir. Qu'est-ce que quelqu'un qui revient nous rendre visite dans une machine à remonter le temps devrait faire attention à ne pas le dire ? C'est ce que je veux étudier ici. Mais je veux faire plus que simplement choquer tout le monde avec l'hérésie du jour. Je veux trouver des recettes générales pour découvrir ce que vous ne pouvez pas dire, à n'importe quelle époque.

Le test conformiste

Commençons par un test : avez-vous des opinions que vous seriez réticent à exprimer devant un groupe de vos pairs ?

Si la réponse est non, vous voudrez peut-être vous arrêter et y réfléchir. Si tout ce que vous croyez est quelque chose que vous êtes censé croire, cela pourrait-il être une coïncidence ? Il y a de fortes chances que ce ne soit pas le cas. Il y a de fortes chances que vous pensiez à ce qu'on vous dit.

L'autre alternative serait que vous ayez examiné de manière indépendante chaque question et que vous ayez trouvé exactement les mêmes réponses qui sont maintenant considérées comme acceptables. Cela semble peu probable, car vous devrez également faire les mêmes erreurs. Les cartographes placent délibérément de légères erreurs dans leurs cartes afin qu'ils puissent savoir quand quelqu'un les copie. Si une autre carte a la même erreur, c'est une preuve très convaincante.

Comme toute autre époque de l'histoire, notre carte morale contient presque certainement des erreurs. Et quiconque commet les mêmes erreurs ne l'a probablement pas fait par accident. Ce serait comme si quelqu'un prétendait avoir décidé indépendamment en 1972 que les jeans pattes d'éléphant étaient une bonne idée.

Si vous croyez tout ce que vous êtes censé faire maintenant, comment pouvez-vous être sûr que vous n'auriez pas également cru tout ce que vous étiez censé croire si vous aviez grandi parmi les propriétaires de plantations de l'avant-guerre civile Sud, ou en Allemagne dans les années 1930 - ou parmi les Mongols en 1200, d'ailleurs ? Il y a de fortes chances que vous l'ayez fait.

À l'ère des termes comme "bien ajusté", l'idée semblait être qu'il y avait quelque chose qui n'allait pas chez vous si vous pensiez des choses que vous n'avez pas osé dire à haute voix. Cela semble à l'envers. Presque certainement, il y a quelque chose qui ne va pas chez vous si vous ne pensez pas des choses que vous n'osez pas dire à haute voix.

Problème

Que ne pouvons-nous pas dire ? Une façon de trouver ces idées est simplement de regarder ce que les gens disent et d'avoir des ennuis [2].

Bien sûr, nous ne cherchons pas seulement des choses que nous ne pouvons pas dire. Nous recherchons des choses que nous ne pouvons pas dire qui sont vraies, ou du moins qui ont suffisamment de chances d'être vraies pour que la question reste ouverte. Mais beaucoup de choses que les gens ont des ennuis pour avoir dit passent probablement ce deuxième seuil inférieur. Personne n'a d'ennuis pour avoir dit que $2 + 2$ est 5, ou que les gens de Pittsburgh mesurent dix pieds de haut. De telles déclarations manifestement fausses pourraient être traitées comme des blagues, ou au pire comme des preuves de folie, mais elles ne sont pas susceptibles de rendre quelqu'un fou. Les déclarations qui rendent les gens fous sont celles qui s'inquiètent qu'on puisse y croire. Je soupçonne que les déclarations qui rendent les gens les plus fous sont celles dont ils craignent qu'elles ne soient vraies.

Si Galilée avait dit que les habitants de Padoue mesuraient dix pieds de haut, il aurait été considéré comme un excentrique inoffensif. Dire que la terre était en orbite autour du soleil était une autre question. L'église savait que cela ferait réfléchir les gens.

Certes, en revenons sur le passé, cette règle empirique fonctionne bien. Beaucoup de déclarations qui ont mis les gens en difficulté semblent faire du mal - moins maintenant. Il est donc probable que les visiteurs de l'avenir soient d'accord avec au moins certaines des déclarations qui mettent les gens en difficulté aujourd'hui. N'avons-nous pas de Galilée ? Peu probable.

Pour les trouver, gardez une trace des opinions qui mettent les gens en difficulté et commencez à demander, cela pourrait-il être vrai ? Ok, c'est

peut-être hérétique (ou n'importe quel équivalent moderne), mais cela pourrait-il aussi être vrai ?

Hérésie

Cependant, cela ne nous obtiendra pas toutes les réponses. Que se passe-t-il si personne n'a encore eu des ennuis pour une idée particulière ? Et si certaines idées seraient si radioactivement controversées que personne n'oserait l'exprimer en public ? Comment pouvons-nous les trouver aussi ?

Une autre approche consiste à suivre ce mot, l'hérésie. Dans chaque détail de l'histoire, il semble qu'il y ait eu des étiquettes qui ont été appliquées aux déclarations pour les abattre avant que quiconque n'ait eu l'occasion de demander si elles étaient vraies ou non. "blasphémie", "sacrilège" et "hérésie" étaient de telles étiquettes pour une bonne partie de l'histoire occidentale, comme l'ont été de plus en plus de temps "indécent", "inapproprié" et "non américain". À présent, ces étiquettes ont perdu leur piqûre. Ils le font toujours. À l'heure actuelle, ils sont principalement utilisés ironiquement. Mais à leur époque, ils avaient une force réelle.

Le mot « défaitiste », par exemple, n'a pas de connotation politique particulière à l'heure actuelle. Mais en Allemagne, en 1917, c'était une arme, utilisée par Ludendorff dans une purge de ceux qui favorisaient une paix négociée. Au début de la Seconde Guerre mondiale, il a été largement utilisé par Churchill et ses partisans pour faire taire leurs adversaires. En 1940, tout argument contre la politique agressive de Churchill était « défaitiste ». Était-ce bien ou mal ? Idéalement, personne n'est assez loin pour demander cela.

Nous avons de telles étiquettes aujourd'hui, bien sûr, beaucoup d'entre elles, de l'"inapproprié" tout usage à la redoutée "divisive". À n'importe quelle période, il devrait être facile de comprendre ce que sont de telles étiquettes, simplement en regardant ce que les gens appellent des idées avec lesquelles ils ne sont pas d'accord en plus de fausses. Lorsqu'un politicien dit que son adversaire se trompe, c'est une critique directe, mais lorsqu'il attaque une déclaration comme "divisive" ou "racialement insensible" au lieu de faire valoir qu'elle est fausse, nous devrions commencer à faire attention.

Donc, une autre façon de comprendre de quels tabous les futures générations se moqueront est de commencer par les étiquettes. Prenez une étiquette - "sexiste", par exemple - et essayez de penser à des idées qui s'appelleraient ainsi. Alors, pour chaque question, est-ce vrai ?

Il suffit de commencer à énumérer des idées au hasard ? Oui, parce qu'ils ne seront pas vraiment aléatoires. Les idées qui viennent à l'esprit en premier seront les plus plausibles. Ce seront des choses que vous avez déjà remarquées, mais que vous ne vous êtes pas laissées penser.

En 1989, des chercheurs intelligents ont suivi les mouvements oculaires des radiologistes alors qu'ils scandaient les images de la poitrine à la recherche de signes de cancer du poumon [3]. Ils ont constaté que même lorsque les radiologistes manquaient une lésion cancéreuse, leurs yeux s'étaient généralement mis en pause sur le site. Une partie de leur cerveau savait qu'il y avait quelque chose là-bas ; cela ne s'est tout simplement pas infiltré dans la connaissance consciente. Je pense que de nombreuses pensées hérétiques intéressantes sont déjà principalement formées dans nos esprits. Si nous désarrêtons temporairement notre autocensure, ce sera le premier à émerger.

Temps et espace

Si nous pouvions nous pencher sur l'avenir, il serait évident de laquelle de nos idées ils riraient. Nous ne pouvons pas le faire, mais nous pouvons faire quelque chose de presque aussi bon : nous pouvons nous pencher sur le passé. Une autre façon de déterminer ce que nous faisons mal est de regarder ce qui était autrefois acceptable et qui est maintenant impensable.

Les changements entre le passé et le présent font parfois des progrès répétitifs. Dans un domaine comme la physique, si nous ne sommes pas d'accord avec les générations passées, c'est parce que nous avons raison et qu'elles ont tort. Mais cela devient de moins en moins vrai à mesure que vous vous éloignez de la certitude des sciences dures. Au moment où vous abordez des questions sociales, de nombreux changements ne sont que de la mode. L'âge du consentement fluctue comme des lignes d'ailes.

Nous pouvons imaginer que nous sommes beaucoup plus intelligents et plus vertueux que les générations précédentes, mais plus vous lisez d'histoire, moins cela semble probable. Dans le passé, les gens nous ressemblaient beaucoup. Pas des héros, pas des barbares. Quelles que soient leurs idées, elles étaient des idées auxquelles les gens raisonnables pouvaient croire.

Voici donc une autre source d'hérésies intéressantes. Différez les idées actuelles par rapport à celles de diverses cultures passées, et voyez ce que vous obtenez [4]. Certaines seront choquantes par rapport aux normes actuelles. Ok, d'accord ; mais qu'est-ce qui pourrait aussi être vrai ?

Vous n'avez pas besoin de regarder dans le passé pour trouver de grandes différences. À notre époque, différentes sociétés ont des idées très différentes de ce qui est correct et de ce qui ne l'est pas. Vous pouvez donc essayer de différer les idées d'autres cultures des nôtres. (La meilleure façon de le faire est de leur rendre visite.)

Vous pourriez trouver des tabous contradictoires. Dans une culture, il peut sembler choquant de penser x, tandis que dans une autre, il serait choquant de ne pas le faire. Mais je pense que le choc est généralement d'un côté. Dans une culture, x est correct, et dans une autre, il est considéré comme choquant. Mon hypothèse est la suivante : que le côté qui est choqué est le plus susceptible d'être celui qui se trompe [5].

Je soupçonne que les seuls tabous qui sont plus que des tabous sont ceux qui sont universels, ou presque. Le meurtre par exemple. Mais toute idée qui est considérée comme inoffensive dans un pourcentage important de temps et de lieux, et qui est pourtant tabou dans la nôtre, est un bon candidat pour quelque chose sur lequel nous nous trompons.

Par exemple, à la haute marque du politiquement correct au début des années 1990, Harvard a distribué à son corps professoral et à son personnel une brochure disant, entre autres, qu'il était inapproprié de complimenter les vêtements d'un collègue ou d'un étudiant. Plus de « belle chemise ». Je pense que ce principe est rare parmi les cultures du monde, passées ou présentes. Il y en a probablement plus là où il est considéré comme particulièrement poli de complimenter les vêtements de quelqu'un que là où il est considéré comme

inapproprié. Il y a donc de fortes chances qu'il s'agisse, sous une forme atténuée, d'un exemple de l'un des tabous qu'un visiteur du futur aurait s'il lui arrivait de mettre sa machine à voyager dans le temps en direction de Cambridge, Massachusetts, 1992.

Puritains

Bien sûr, s'ils ont des machines à remonter le temps à l'avenir, ils auront probablement un manuel de référence séparé juste pour Cambridge. Cela a toujours été un endroit difficile, une ville de pointilleux et de méchants, où vous êtes susceptible de faire corriger à la fois votre grammaire et vos idées dans la même conversation. Et cela suggère une autre façon de trouver des tabous. Cherchez des puritains et voyez ce qu'ils ont dans la tête.

Les têtes d'enfants sont les dépôts de tous nos tabous. Il nous semble approprié que les idées des enfants soient brillantes et propres. L'image que nous leur donnons du monde n'est pas simplement simplifiée, pour s'adapter à leur esprit en développement, mais aussi désinfectée, pour s'adapter à nos idées sur ce que les enfants devraient penser [6].

Vous pouvez le voir à petite échelle en matière de "gros mots". Beaucoup de mes amis commencent à avoir des enfants maintenant, et ils essaient tous de ne pas utiliser de mots comme "putain" et "merde" avec le bébé, de crainte que bébé ne commence à utiliser ces mots aussi. Mais ces mots font partie du langage, et les adultes les utilisent tout le temps. Les parents donnent donc à leurs enfants une idée inexacte de la langue en ne les utilisant pas. Pourquoi font-ils cela ? Parce qu'ils ne pensent pas qu'il soit approprié que les enfants utilisent toute la langue. Nous aimons que les enfants paraissent innocents [7].

De même, la plupart des adultes donnent délibérément aux enfants une vision trompeuse du monde. L'un des exemples les plus évidents est le Père Noël. Nous pensons que c'est mignon pour les petits enfants de croire au Père Noël. Je pense moi-même que c'est mignon pour les petits enfants de croire au Père Noël. Mais on se demande, est-ce qu'on leur dit ces choses pour eux, ou pour nous ?

Je ne plaide pas pour ou contre cette idée ici. Il est probablement inévitable que les parents veuillent habiller l'esprit de leurs enfants dans de jolies petites tenues de bébé. Je le ferai probablement moi-même. La chose importante pour nos besoins est que, par conséquent, le cerveau d'un adolescent bien élevé est une collection plus ou moins complète de tous nos tabous - et en parfait état, parce qu'ils ne sont pas entachés par l'expérience. Quoi que nous pensions que cela s'en sortira plus tard ridicule, c'est presque certainement à l'intérieur de cette tête.

Comment pouvons-nous arriver à ces idées ? Par l'ex-expert de pensée suivant. Imaginez une sorte de personnage de Conrad des derniers jours qui a travaillé pendant un certain temps comme mercenaire en Afrique, pendant un certain temps comme médecin au Népal, pendant un certain temps comme directeur d'une boîte de nuit à Miami. Les détails n'ont pas d'importance, juste quelqu'un qui a vu beaucoup de choses. Maintenant, imaginez comparer ce qu'il y a à l'intérieur de la tête de ce type avec ce qu'il y a à l'intérieur de la tête d'une jeune fille de seize ans de la banlieue. Qu'est-ce qui, selon lui, la choquerait ? Il connaît le monde ; elle connaît, ou du moins incarne, les tabous présents. Soustrayez l'un de l'autre, et le résultat est ce que nous ne pouvons pas dire.

Mécanisme

Je peux penser à une autre façon de comprendre ce que nous ne pouvons pas dire : regarder comment les tabous sont créés. Comment les modes morales surgissent-elles et pourquoi sont-elles adoptées ? Si nous pouvons comprendre ce mécanisme, nous pourrons peut-être le voir à l'œuvre à notre époque.

Les modes morales ne semblent pas être créées comme le sont les modes ordinaires. Les modes ordinaires semblent surgir par accident lorsque tout le monde imite le caprice d'une personne influente. La mode pour les chaussures à larges orteils dans l'Europe de la fin du XVe siècle a commencé parce que Charles VIII de France avait six orteils sur un pied. La mode pour le nom Gary a commencé lorsque l'acteur Frank Cooper a adopté le nom d'une ville de moulin difficile en Indiana. Les modes morales semblent plus souvent être créées délibérément. Quand il y a quelque chose que nous ne pouvons pas dire, c'est souvent parce qu'un groupe ne veut pas que nous le fassions.

L'interdiction sera plus forte lorsque le groupe sera nerveux. L'ironie de la situation de Galilée était qu'il a eu des ennuis pour avoir répéter les idées de Copernic. Copernic lui-même ne l'a pas fait. En fait, Copernic était le chanoine d'une cathédrale et dédia son livre au pape. Mais à l'époque de Galilée, l'église était en proie à la Contre-Réforme et était beaucoup plus préoccupée par les idées peu orthodoxes.

Pour lancer un tabou, un groupe doit être à mi-chemin entre la faiblesse et le pouvoir. Un groupe confiant n'a pas besoin de tabous pour le protéger. Il n'est pas considéré comme inapproprié de faire des marques désobligeantes à propos des Américains ou des Anglais. Et pourtant, un groupe doit être assez puissant pour imposer un tabou. Les coprophiles, au moment d'écrire ces lignes, ne semblent pas être assez nombreux ou énergiques pour avoir fait promouvoir leurs intérêts dans un style de vie.

Je soupçonne que la plus grande source de tabous moraux va être des luttes de pouvoir dans lesquelles un côté a à peine le dessus. C'est là que vous trouverez un groupe assez puissant pour imposer les huées, mais assez faible pour en avoir besoin.

La plupart des luttes, quelles qu'elles soient, seront jouées comme des luttes entre des idées concurrentes. La Réforme anglaise était au fond une lutte pour la richesse et le pouvoir, mais elle a fini par être comme une lutte pour préserver les âmes des Anglais de l'influence corrompue de Rome. Il est plus facile d'amener les gens à se battre pour une idée. Et quel que soit le camp qui gagnera, leurs idées seront également considérées comme ayant triomphé, comme si Dieu voulait signaler son accord en choisissant ce camp comme vainqueur.

Nous aimons souvent penser à la Seconde Guerre mondiale comme un triomphe de la liberté sur le totalitarisme. Nous oublions commodément que l'Union soviétique a également été l'un des gagnants.

Je ne dis pas que les luttes ne sont jamais une question d'idées, juste qu'elles seront toujours faites pour sembler être une question d'idées, qu'elles le soient ou non. Et tout comme il n'y a rien d'autant démodé que la dernière mode abandonnée, il n'y a rien d'autant mal que les principes de l'adversaire le plus

récemment vaincu. L'art de la représentation ne se remet que de l'approbation d'Hitler et de Staline [8].

Bien que les modes dans les idées aient tendance à provenir de sources différentes de celles des modes dans les vêtements, le mécanisme de leur adoption semble à peu près le même. Les premiers adoptants seront motivés par l'ambition : des gens consciemment cool qui veulent se distinguer de l'élevage commun. Au fur et à mesure que la mode s'établira, ils seront rejoints par un deuxième groupe, beaucoup plus grand, poussé par la peur [9]. Ce deuxième groupe adopte la mode non pas parce qu'il veut se démarquer, mais parce qu'il a peur de se démarquer.

Donc, si vous voulez comprendre ce que nous ne pouvons pas dire, regardez la machinerie de la mode et essayez de prédire ce qu'elle rendrait impossible à dire. Quels groupes sont puissants mais nerveux, et quelles idées aimeraient-ils supprimer ? Quelles idées ont été ternies par l'association lorsqu'elles se sont retrouvées du côté perdant d'une lutte récente ? Si une personne consciemment cool voulait se différencier des modes précédentes (par exemple de ses parents), laquelle de leurs idées aurait-elle tendance à rejeter ? Qu'est-ce que les gens à l'esprit conventionnel ont peur de dire ?

Cette technique ne nous trouvera pas toutes les choses que nous ne pouvons pas dire. Je peux penser à certains qui ne sont pas le résultat d'une lutte récente. Beaucoup de nos tabous sont profondément enracinés dans le passé. Mais cette approche, combinée aux quatre précédentes, fera repartir un bon nombre d'idées impensables.

Pourquoi

Certains se demanderaient, pourquoi voudrait-on faire cela ? Pourquoi fouiller délibérément parmi des idées méchantes et peu réputées ? Pourquoi regarder sous les rochers ?

Je le fais, tout d'abord, pour la même raison que j'ai regardé sous les rochers quand j'étais enfant : la pure curiosité. Et je suis particulièrement curieux de tout ce qui est interdit. Laissez-moi voir et décider par moi-même.

Deuxièmement, je le fais parce que je n'aime pas l'idée de me tromper. Si, comme d'autres époques, nous croyons des choses qui sembleront plus tard ridicules, je veux savoir ce qu'elles sont afin que, au moins, je puisse éviter de les croire.

Troisièmement, je le fais parce que c'est bon pour le cerveau. Pour faire du bon travail, vous avez besoin d'un cerveau qui peut aller n'importe où. Et vous avez surtout besoin d'un cerveau qui a l'habitude d'aller là où il n'est pas censé aller.

Le grand travail a tendance à sortir d'idées que d'autres ont trop regardées, et aucune idée n'est aussi négligée qu'une idée impensable. Sélection naturelle, par exemple. C'est si simple. Pourquoi personne n'y a-t-il pensé auparavant ? Eh bien, c'est trop évident. Darwin lui-même a fait attention à contourner les implications de sa théorie. Il voulait passer son temps à réfléchir à la biologie, et non à se disputer avec des gens qui l'accusaient d'être athée.

Dans les sciences, en particulier, c'est un grand avantage de pouvoir remettre en question les hypothèses. Le mode opératoire des scientifiques, ou du moins des bons, est précisément cela : cherchez des endroits où la sagesse conventionnelle est brisée, puis essayez de séparer les fissures et de voir ce qu'il y a en dessous. C'est de là que viennent les nouvelles théories.

En d'autres termes, un bon scientifique n'ignore pas seulement la sagesse traditionnelle, mais fait un effort particulier pour la briser. Les scientifiques cherchent des ennuis. Cela devrait être le mode opératoire de n'importe quel érudit, mais les scientifiques semblent beaucoup plus disposés à regarder sous les rochers.

Pourquoi ? Il se pourrait que les scientifiques soient tout simplement plus intelligents ; la plupart des physiciens pourraient, si nécessaire, passer par un programme de doctorat en littérature française, mais peu de professeurs de littérature française pourraient passer par un programme de doctorat en physique [10]. Ou cela pourrait être - parce qu'il est plus clair dans les sciences si les théories sont vraies ou fausses, et cela rend les scientifiques plus audacieux. (Ou il se pourrait que, parce qu'il est plus clair dans les sciences si les théories sont vraies ou fausses, vous devez être intelligent pour obtenir un emploi en tant que scientifique, plutôt que simplement un bon politicien.)

Quelle qu'en soit la raison, il semble y avoir une corrélation claire entre l'intelligence et la volonté d'envisager des idées choquantes. Ce n'est pas seulement parce que des personnes intelligentes travaillent activement pour trouver des trous dans la pensée conventionnelle. Les conventions ont également moins d'emprise sur elles au départ. Vous pouvez le voir dans la façon dont ils s'habillent.

Ce n'est pas seulement dans les sciences que l'hérésie est payante. Dans n'importe quel domaine de compétition, vous pouvez gagner gros en voyant des choses que les autres n'aiment pas. Et dans tous les domaines, il y a probablement des hérésies que peu osent prononcer. Au sein de l'industrie automobile. L'industrie automobile américaine se préoccupe beaucoup de la baisse des parts de marché. Pourtant, la cause est si évidente que tout observateur extérieur pourrait l'expliquer en une seconde : ils fabriquent de mauvaises voitures. Et ils l'ont depuis si longtemps que, maintenant, les marques de voitures américaines sont des anti marques - quelque chose que vous achèteriez une voiture malgré, pas à cause de. Cadillac a cessé d'être la Cadillac des voitures vers 1970. Et pourtant, je soupçonne que personne n'ose le dire [11]. Sinon, ces entreprises auraient essayé de résoudre le problème.

S'entraîner à penser à des pensées impensables a des avantages au-delà des pensées elles-mêmes. C'est comme s'étirer. Lorsque vous vous étirez avant de courir, vous placez votre corps dans des positions beaucoup plus extrêmes que tout ce qu'il supposera pendant la course. Si vous pouvez penser des choses tellement hors des sentiers battus qu'elles rendraient les cheveux des gens debout, vous n'aurez aucun problème avec les petits voyages hors des sentiers battus que les gens appellent innovants.

Pensieri Stretti

Quand vous trouvez quelque chose que vous ne pouvez pas dire, qu'en faites-vous ? Mon conseil est de ne pas le dire. Ou du moins, choisissez vos batailles.

Supposons qu'à l'avenir, il y ait un mouvement pour interdire la couleur jaune. Les propositions de peindre quoi que ce soit en jaune sont dénoncées

comme « yellowist », tout comme toute personne soupçonnée d'aimer la couleur. Les gens qui aiment l'orange sont tolérés mais considérés avec suspicion. Supposons que vous réalisiez il n'y a rien de mal avec le jaune. Si vous faites le tour en disant cela, vous serez également dénoncé comme un yellowist, et vous vous retrouverez à avoir beaucoup de disputes avec les anti-jaunâtres. Si votre objectif dans la vie est de réhabiliter la couleur jaune, c'est peut-être ce que vous voulez. Mais si vous êtes surtout intéressé par d'autres questions, être étiqueté comme un yellowist ne sera qu'une distraction. Discutez avec des idiots, et vous devenez un idiot.

La chose la plus importante est de pouvoir penser ce que vous voulez, pas de dire ce que vous voulez. Et si vous sentez que vous devez dire tout ce que vous pensez, cela peut vous empêcher de penser à des pensées inappropriées. Je pense qu'il est préférable de suivre la politique opposée. Tracez une ligne nette entre vos pensées et votre discours. Dans votre tête, tout est permis. Dans ma tête, je mets un point d'honneur à encourager les pensées les plus scandaleuses que je puisse imaginer. Mais, comme dans une société secrète, rien de ce qui se passe à l'intérieur du bâtiment ne doit être dit à des étrangers. La première règle de Fight Club est que vous ne parlez pas de Fight Club.

Lorsque Milton allait visiter l'Italie dans les années 1630, Sir Henry Wootton, qui avait été ambassadeur à Venise, lui a dit que sa devise devrait être "i pensieri stretti & il viso sciolto". Des pensées fermées et un visage ouvert. Souriez à tout le monde et ne leur dites pas ce que vous pensez. C'était un sage conseil. Milton était un homme argumentatif, et l'Inquisition était un peu agitée à ce moment-là. Mais la différence entre la situation de Milton et la nôtre n'est qu'une question de degré. Chaque époque a ses hérésies, et si vous n'êtes pas emprisonné pour eux, vous aurez au moins assez d'ennuis pour que cela devienne une distraction complète.

J'avoue qu'il semble lâche de se taire. Quand j'ai lu sur le harcèlement auquel les scientologues soumettent leurs critiques [12] ou des personnes marquées comme antisémites pour s'être prononcées contre les violations des droits de l'homme israéliens [13] ou des chercheurs menacés de poursuites en vertu de la loi DMCA [14], une partie de moi a envie de dire : "D'accord, bande de salauds, allez-y". Le problème, c'est qu'il y a tellement de choses qu'on ne

peut pas dire. Si vous les disiez tous, vous n'auriez plus de temps pour votre vrai travail. Vous auriez à vous transformer en Noam Chomsky [15].

Le problème à garder vos pensées secrètes, cependant, est que vous perdez les avantages de la discussion. Parler d'une idée conduit à plus d'idées. Donc, le plan optimal, si vous pouvez le gérer, est d'avoir quelques amis de confiance à qui vous pouvez parler ouvertement. Ce n'est pas seulement une façon de développer des idées ; c'est aussi une bonne règle de base pour choisir des amis. Les personnes à qui vous pouvez dire des choses hérétiques sans vous faire sauter dessus sont également les plus intéressantes à connaître.

Viso Sciolto ?

La meilleure politique est peut-être de préciser que vous n'êtes pas d'accord avec le fanatisme actuel de votre époque, mais de ne pas être trop précis sur ce avec quoi vous n'êtes pas d'accord. Les fanatiques vont essayer de vous faire sortir, mais vous n'avez pas à y répondre. S'ils essaient de vous forcer à traiter une question selon leurs conditions en demandant « êtes-vous avec nous ou contre nous ? » Vous pouvez toujours simplement répondre « ni l'un ni l'autre ».

Mieux encore, répondez « Je n'ai pas décidé ». C'est ce que Larry Summers a fait lorsqu'un groupe a essayé de le mettre dans cette position [16]. En s'expliquant plus tard, il a dit "Je ne fais pas de tests de tournesol". Beaucoup de questions sur lesquelles les gens se posent sont en fait assez compliquées. Il n'y a pas de prix pour obtenir la réponse rapidement.

Si les anti-jaunâtres semblent devenir incontrôlables et que vous voulez riposter, il existe des moyens de le faire sans vous faire accuser de yellowisme. Comme les tirailleurs d'une ancienne armée, vous voulez éviter d'engager directement le corps principal des troupes de l'ennemi. Mieux vaut les harceler avec des flèches à distance.

Une façon de le faire est de faire monter le débat d'un niveau d'abstraction. Si vous vous opposez à la censure en général, vous pouvez éviter d'être accusé de toute hérésie contenue dans le livre ou le film que quelqu'un essaie de censurer. Vous pouvez attaquer les étiquettes avec des métal'étiquettes :

des étiquettes qui font référence à l'utilisation d'étiquettes pour empêcher la discussion. La propagation du terme « politiquement correct » signifiait le début de la fin du politiquement correct, car il permettait d'attaquer le phénomène dans son ensemble sans être accusé d'aucune des hérésies spécifiques qu'il cherchait à supprimer.

Une autre façon de contre-attaquer est avec la métaphore. Arthur Miller a sapé le Comité des activités non américaines de la Chambre en écrivant une pièce de théâtre, *The Crucible*, sur les procès des sorcières de Salem. Il n'a jamais renvoyé directement au comité et ne leur a donc donné aucun moyen de répondre. Que pourrait faire HUAC, défendre les procès des sorcières de Salem ? Et pourtant, la métaphore de Miller est si bien restée qu'à ce jour, les activités du comité sont souvent décrites comme une « chasse aux sorcières ».

Le meilleur de tous, c'est probablement l'humour. Les fanatiques, quelle que soit leur cause, manquent invariablement de sens de l'humour. Ils ne peuvent pas répondre en nature aux blagues. Ils sont aussi malheureux sur le territoire de l'humour qu'un chevalier à cheval sur une patinoire. La pudeur victorienne, par exemple, semble avoir été vaincue principalement en la traitant comme une blague. De même, sa réincarnation en tant que politiquement correct. « Je suis heureux d'avoir réussi à écrire *The Crucible* », a écrit Arthur Miller, « mais en regardant en arrière, j'ai souvent souhaité avoir le tempérament de faire une comédie absurde, ce que la situation méritait. » [17].

Toujours Interroger

Un ami néerlandais dit que je devrais utiliser la Hollande comme exemple d'une société tolérante. Il est vrai qu'ils ont une longue tradition d'ouverture d'esprit comparative. Pendant des siècles, les Pays-Bas ont été l'endroit où aller pour dire des choses que vous ne pouviez dire nulle part ailleurs, et cela a contribué à faire de la région un centre d'érudition et d'industrie (qui sont étroitement liés depuis plus longtemps que la plupart des gens ne le pensent). Descartes, bien que revendiqué par les Français, a fait une grande partie de sa pensée en Hollande.

Et pourtant, je me demande. Les Néerlandais semblent vivre jusqu'au cou dans les règles et règlements. Il y a tellement de choses que vous ne pouvez pas faire là-bas ; n'y a-t-il vraiment rien que vous ne puissiez pas dire ?

Certes, le fait qu'ils valorisent l'ouverture d'esprit n'est pas une garantie. Qui pense qu'ils ne sont pas ouverts d'esprit ? Notre hypothétique prim miss de la banlieue pense qu'elle est ouverte d'esprit. Ne lui a-t-on pas appris à l'être ? Demandez à n'importe qui, et ils diront la même chose : ils sont assez ouverts d'esprit, bien qu'ils traduisent la ligne sur les choses qui sont vraiment fausses [18]. En d'autres termes, tout va bien, sauf les choses qui ne le sont pas.

Quand les gens sont mauvais en mathématiques, ils le savent, parce qu'ils obtiennent de mauvaises réponses aux tests. Mais quand les gens sont mauvais dans l'ouverture d'esprit, ils ne le savent pas. En fait, ils ont tendance à penser le contraire. N'oubliez pas que c'est la nature de la mode d'être invisible. Cela ne fonctionnerait pas autrement. La mode ne semble pas être à la mode pour quelqu'un qui en est imprégné. Cela semble juste être la bonne chose à faire. Ce n'est qu'en regardant de loin que nous voyons des oscillations dans l'idée des gens de la bonne chose à faire, et que nous pouvons les identifier comme des modes.

Le temps nous donne une telle distance gratuitement. En effet, l'arrivée de nouvelles modes rend les vieilles modes faciles à voir, parce qu'elles semblent si ridicules par contre. D'une extrémité de la balançoire d'un pendule, l'autre extrémité semble particulièrement éloignée.

Pour voir la mode à votre époque, cependant, il faut un effort conscient. Sans avoir le temps de vous donner de la distance, vous devez créer vous-même une distance. Au lieu de faire partie de la foule, tenez-vous aussi loin que possible et regardez ce qu'elle fait. Et faites particulièrement attention chaque fois qu'une idée est supprimée. Les filtres Web pour les enfants et les employés interdisent souvent les sites contenant de la pornographie, de la violence et des discours de haine. Qu'est-ce qui compte comme pornographie et violence ? Et qu'est-ce que, exactement, « le discours de haine ? » Cela ressemble à une phrase de 1984.

Des étiquettes comme celle-ci sont probablement le plus grand indice externe. Si une déclaration est fausse, c'est la pire chose que vous puissiez dire à ce sujet. Vous n'avez pas besoin de dire que c'est hérétique. Et si ce n'est pas faux, il ne devrait pas être supprimé. Ainsi, lorsque vous voyez des déclarations attaquées en tant que x-ist ou y-ic (substituez vos valeurs actuelles de x et y), que ce soit en 1630 ou en 2030, c'est un signe certain que quelque chose ne va pas. Lorsque vous entendez de telles étiquettes être utilisées, demandez pourquoi.

Surtout si vous vous entendez les utiliser. Ce n'est pas seulement la foule que vous devez apprendre à regarder à distance. Vous devez être capable de regarder vos propres pensées à distance. Ce n'est pas une idée radicale, soit dit en passant, c'est la principale différence entre les enfants et les adultes. Quand un enfant se met en colère parce qu'il est fatigué, il ne le fait pas savoir ce qui se passe. Un adulte peut se distancer suffisamment de la situation pour dire "peu importe, je suis juste fatigué". Je ne vois pas pourquoi on ne pourrait pas, par un processus similaire, apprendre à reconnaître et à écarter les effets des modes morales.

Vous devez prendre cette mesure supplémentaire si vous voulez penser clairement. Mais c'est plus difficile, parce que maintenant vous travaillez contre les coutumes sociales plutôt qu'avec elles. Tout le monde vous encourage à grandir au point où vous pouvez réduire votre propre mauvaise humeur. Peu vous encouragent à continuer au point où vous pouvez écarter les mauvaises humeurs de la société.

Comment pouvez-vous voir la vague, quand vous êtes l'eau ? Toujours poser des questions. C'est la seule défense. Que pouvez-vous dire ? Et pourquoi ?

Chapitre 4

Bonne Mauvaise Attitude

Pour la presse populaire, un hacker signifie quelqu'un qui fait irruption dans les ordinateurs. Chez les programmeurs, cela signifie un bon programmeur. Mais les deux significations sont liées. Pour programmer, "hacker" implique la maîtrise dans le sens le plus littéral du terme : quelqu'un qui peut faire en sorte qu'un ordinateur fasse ce qu'il veut, que l'ordinateur le veuille ou non.

Pour ajouter à la confusion, le nom « hack » a également deux sens. Cela peut être un compliment ou une insulte. C'est ce qu'on appelle un hack quand on fait quelque chose d'une manière peu élégante. Mais quand vous faites quelque chose de si intelligent que vous battez d'une manière ou d'une autre le système, cela s'appelle aussi un hack. Le mot est utilisé plus souvent dans le premier sens que dans le second sens, probablement parce que les solutions laides sont plus courantes que les solutions brillantes.

Croyez-le ou non, les deux sens de « hack » sont également liés. Les solutions laides et imaginatives ont quelque chose en commun : elles enfreignent toutes les deux les règles. Et il y a un continuum progressif entre la violation des règles qui est simplement repoussante (utiliser du ruban adhésif pour attacher quelque chose à votre vélo) et la violation des règles qui est brillamment imaginative (déjeter l'espace euclidien).

Le bidouillage est antérieur aux ordinateurs. Lorsqu'il travaillait sur le projet Manhattan, Richard Feynman avait l'habitude de s'amuser en pénétrant dans des coffres-forts contenant des documents secrets. Cette tradition se poursuit aujourd'hui. Lorsque nous étions à l'école supérieure, un de mes amis pirates qui a passé trop de temps autour du MIT avait son propre kit de sélection de serrures [1]. (Il dirige maintenant un fonds spéculatif, une entreprise non liée.)

Il est parfois difficile d'expliquer aux autorités pourquoi on voudrait faire de telles choses. Un autre de mes amis s'est un jour retrouvé dans le pétrin avec le gouvernement pour avoir pénétré par effraction dans les ordinateurs. Ce n'est que récemment qu'ils ont été déclarés comme un crime, et le FBI a constaté que leur technique d'enquête habituelle ne fonctionnait pas. L'enquête policière

commence par un motif. Les motifs habituels sont peu nombreux : drogue, argent, sexe, vengeance. La curiosité intellectuelle n'était pas l'un des motifs sur la liste du FBI. En effet, tout le concept leur semblait étranger.

Ceux qui sont en autorité ont tendance à être agacés par l'attitude générale de désobéissance des pirates informatiques. Mais cette désobéissance est un sous-produit des qualités qui font d'eux de bons programmeurs. Ils peuvent se moquer du PDG lorsqu'il parle dans un nouveau discours d'entreprise générique, mais ils se moquent aussi de quelqu'un qui leur dit qu'un certain problème ne peut pas être résolu. Supprimez l'un, et vous supprimez l'autre.

Cette attitude est parfois affectée. Parfois, les jeunes programmeurs remarquent les excentricités d'éminents hackers informatiques et décident d'adopter certains de leurs afin de paraître plus intelligents. La fausse version n'est pas simplement ennuyeuse ; l'attitude piquante de ces poseurs peut en fait ralentir le processus d'innovation.

Mais même en tenant compte de leurs excentricités agaçantes, l'attitude désobéissante des hackers est un gain net. J'aimerais que ses avantages soient mieux compris.

Par exemple, je soupçonne que les gens d'Hollywood sont simplement mystifiés par les attitudes des hackers à l'égard des droits d'auteur. Ils sont un sujet éternel de discussion animée sur Slashdot. Mais pourquoi les gens qui programment des ordinateurs devraient-ils être si préoccupés par les droits d'auteur, de toutes choses ?

En partie parce que certaines entreprises utilisent des *mécanismes* pour empêcher la copie. Montrez à n'importe quel hacker un verrou et sa première idée sera de le crocheter. Mais il y a une raison plus profonde pour laquelle les hackers sont alarmés par des raisons telles que les droits d'auteur et les brevets. Ils considèrent les mesures de plus en plus agressives pour protéger la "propriété intellectuelle" comme une menace pour la liberté intellectuelle dont ils ont besoin pour faire leur travail. Et ils ont raison.

C'est en parcourant l'intérieur de la technologie actuelle que les hackers ont des idées pour la prochaine génération. Non merci, les propriétaires

intellectuels peuvent dire que nous n'avons pas besoin d'aide extérieure. Mais ils ont tort. La prochaine génération de technologie informatique a souvent - peut-être plus souvent qu'autrement - été développée par des étrangers. En 1977, il n'y avait pas de doutes au sein d'IBM développer ce qu'il s'attendait à être la prochaine génération d'ordinateurs d'entreprise. Ils se sont trompés. La prochaine génération d'ordinateurs d'affaires était en train d'être développée sur des lignes entièrement différentes par deux gars aux cheveux longs appelés Steve dans un garage à Los Altos. À peu près au même moment, les pouvoirs coopéraient pour développer le système d'exploitation officiel de la prochaine génération, Multics. Mais deux gars qui pensaient que Multics était trop complexe sont partis et ont écrit le leur. Ils lui ont donné un nom qui était une référence plaisante à Multics : Unix.



Jobs et Wozniak avec un dispositif de contournement, 1975.

Les dernières lois sur la propriété intellectuelle imposent des restrictions sans précédent sur le type de fouille qui mène à de nouvelles idées. Dans le passé, un concurrent pouvait utiliser des brevets pour vous empêcher de vendre une copie de quelque chose qu'il a fabriqué, mais il ne pouvait pas vous empêcher d'en démonter un pour voir comment cela fonctionnait. Les dernières lois en font un crime. Comment allons-nous développer une nouvelle technologie si nous ne pouvons pas étudier la technologie actuelle pour comprendre comment l'améliorer ?

Ironiquement, les hackers ont provoqué cela sur eux-mêmes. Les ordinateurs sont les responsables du problème. Les systèmes de contrôle à

l'intérieur des machines étaient auparavant physiques : engrenages, leviers et cames. De plus en plus, le cerveau (et donc la valeur) des produits est dans le logiciel [2]. Et par là, je veux dire le logiciel au sens général : c'est-à-dire les données. La chanson d'un disque compact est physiquement gravée sur le plastique. Une chanson sur le disque d'un iPod est simplement stockée dessus.

Les données sont par définition faciles à copier. Et Internet rend les copies faciles à distribuer. Il n'est donc pas étonnant que les entreprises aient peur. Mais, comme cela arrive souvent, la peur a assombri leur jugement. Le gouvernement a réagi par des lois draconiennes pour protéger les biens de l'intelligence. Ils ont probablement de bons moyens. Mais ils ne se rendent peut-être pas compte que de telles lois feraient plus de mal que de bien.

Pourquoi les programmeurs s'opposent-ils si violemment à ces lois ? Si j'étais législateur, je serais intéressé par ce mystère - pour la même raison que, si j'étais agriculteur et que j'entendais soudainement beaucoup de chamaïlle venant de mon berger de poule un soir, je voudrais sortir et enquêter. Les hackers ne sont pas stupides, et l'unanimité est très rare dans ce monde. Donc, s'ils crient tous, il y a peut-être quelque chose qui ne va pas.

Se pourrait-il que de telles lois, bien que destinées à protéger les Etats-Unis, lui nuiraient réellement ? Pensez-y. Il y a quelque chose de très américain dans le fait que Feynman entre par effraction dans des coffres-forts pendant le projet Manhattan. Il est difficile d'imaginer que les autorités aient un sens de l'humour à propos de telles choses en Allemagne à ce moment-là. Ce n'est peut-être pas une coïncidence.

Les hackers sont indisciplinés. C'est l'essence du bidouillage informatique. Et c'est aussi l'essence de l'Amérique. Ce n'est pas un hasard si la Silicon Valley se trouve en Amérique, et non en France, ni en Allemagne, ni en Angleterre, ni au Japon. Dans ces pays, les gens colorent à l'intérieur des lignes.

J'ai vécu pendant un certain temps à Florence. Mais après avoir été là quelques mois, j'ai réalisé que ce que j'espérais inconsciemment trouver était de retourner à l'endroit que je venais de quitter. La raison pour laquelle Florence est célèbre est qu'en 1450, c'était New York. En 1450, il était rempli du genre de

gens turbulents et ambitieux que vous trouvez maintenant aux Etats-Unis. (Je suis donc retourné aux Etats-Unis.)

C'est en grande partie à l'avantage des Etats-Unis qu'il s'agisse d'une sphère sympathique pour le bon type de indiscipliné - qu'il s'agit d'une maison non seulement pour les intelligents, mais aussi pour les "smart-alecks. Et les hackers sont invariablement des smart-alecks. Si nous avions une fête nationale, ce serait le 1er avril. Cela en dit long sur notre travail que nous utilisons le même mot pour une solution brillante ou terriblement ringarde. Lorsque nous en cuisinons un, nous ne sommes pas toujours sûrs à 100 % de quel type il s'agit. Mais tant qu'il a le bon type de tort, c'est un signe prometteur. Il est étrange que les gens considèrent la programmation comme précise et méthodique. Les ordinateurs sont précis et méthodiques. Le bidouillage informatique est quelque chose que vous faites avec un rire joyeux.

Dans notre monde, certaines des solutions les plus caractéristiques ne sont pas loin des blagues pratiques. IBM a sans aucun doute été plutôt surpris par les conséquences de l'accord de licence pour DOS, tout comme l'hypothétique "adversaire" doit l'être lorsque Michael Rabin résout un problème en le redéfinissant comme un problème plus facile à résoudre.

Les smart-alecks doivent avoir une idée aiguë de ce qu'ils peuvent faire. Et ces derniers temps, les hackers ont senti un changement dans l'atmosphère. Ces derniers temps, le bidouillage semble plutôt mal vu.

Pour les hackers, la récente contraction des libertés civiles semble particulièrement inquiétante. Les personnes extérieures ne doivent pas s'en douter non plus. Pourquoi devrions-nous nous soucier des libertés civiles ? Pourquoi les programmeurs, plus que les dentistes, les vendeurs ou les paysagistes ?

Permettez-moi de mettre l'affaire en termes qu'un fonctionnaire du gouvernement apprécierait. Les libertés civiles ne sont pas seulement un ornement, ou une tradition américaine pittoresque. Les libertés civiles rendent les pays riches. Si vous faisiez un graphique du PNB par habitant par rapport aux libertés civiles, vous remarqueriez une tendance certaine. Les libertés

civiles pourraient-elles vraiment être une cause, plutôt qu'un simple effet ? Je pense que c'est le cas. Je pense qu'une société dans laquelle les gens peuvent faire et dire ce qu'ils veulent aura également tendance à être une société dans laquelle les solutions les plus efficaces gagnent, plutôt que celles parrainées par les personnes les plus influentes. Les pays autoritaires deviennent corrompus, les pays corrompus deviennent pauvres et les pays pauvres sont faibles. Il me semble qu'il y a une courbe de Laffer pour le pouvoir gouvernemental, tout comme pour les renouvellements fiscaux [3]. Au moins, il semble assez probable qu'il serait stupide d'essayer l'expérience et de la découvrir. Contrairement aux taux d'imposition élevés, vous ne pouvez pas abroger le totalitarisme si cela s'avère être une erreur.

C'est pourquoi les hackers s'inquiètent. Le gouvernement qui espionne les gens n'a pas littéralement obligé les programmeurs à écrire un pire code. Cela mène finalement à un monde dans lequel les mauvaises idées gagneront. Et parce que c'est si important pour les hackers, ils y sont particulièrement sensibles. Ils peuvent sentir le totalitarisme qui s'approche de loin, comme les animaux peuvent sentir un orage qui s'approche.

Il serait ironique si, comme le craignent les hackers, les mesures récentes visant à protéger la sécurité nationale et la propriété intellectuelle soient un missile visant directement ce qui fait le succès des Etats-Unis. Mais ce ne serait pas la première fois que les mesures prises dans une atmosphère de panique avaient le contraire de l'effet escompté.

Il y a quelque chose comme l'américanité. Il n'y a rien de tel que de vivre à l'étranger pour vous apprendre cela. Et si vous voulez savoir si quelque chose va nourrir ou écraser cette qualité, il serait difficile de trouver un meilleur groupe de discussion que les hackers, parce qu'ils sont les plus proches de tous les groupes que je connais pour l'incarner. Plus proche, probablement, que les hommes qui dirigent notre gouvernement, qui, pour tout leur discours sur le patriotisme, me rappellent plus Richelieu ou Mazarin que Thomas Jefferson ou George Washington.

Lorsque vous lisez ce que les pères fondateurs avaient à dire pour eux-mêmes, ils ressemblent plus à des hackers. « L'esprit de résistance au

gouvernement », a écrit Jefferson, « est si précieux à certaines occasions, que je souhaite qu'il soit toujours maintenu en vie. »

Imaginez qu'un président américain dise cela aujourd'hui. Comme les remarques d'une vieille grand-mère franche, les paroles des pères fondateurs ont embarrassé des générations de leurs successeurs moins fiables. Ils nous rappellent d'où nous venons. Ils nous rappellent que ce sont les gens qui enfreignent les règles qui sont la source de la richesse et du pouvoir des Etats-Unis.

Ceux qui sont en mesure d'imposer des règles veulent naturellement qu'elles soient respectées. Mais faites attention à ce que vous demandez. Vous pourriez l'obtenir.

Chapitre 5

L'Autre Voie à Suivre

À l'été 1995, mon ami Robert Morris et moi avons décidé de démarrer une start-up. La campagne de relations publiques qui a conduit à l'introduction en bourse de Netscape battait son plein à l'époque, et la presse a beaucoup parlé du commerce en ligne. À l'époque, il y avait peut-être trente magasins réels sur le Web, tous fabriqués à la main. S'il devait y avoir beaucoup de magasins en ligne, il faudrait des logiciels pour les fabriquer, alors nous avons décidé d'en écrire.

Pour la première semaine environ, nous avions l'intention d'en faire une application de bureau ordinaire. Puis un jour, nous avons eu l'idée de faire fonctionner le logiciel sur notre serveur web, en utilisant le navigateur comme interface. Nous avons essayé de réécrire le logiciel pour qu'il fonctionne sur le Web, et il était clair que c'était la voie à prendre. Si nous écrivions notre logiciel pour l'exécuter sur le serveur, ce serait beaucoup plus facile pour les utilisateurs et pour nous aussi.

Cela s'est avéré être un bon plan. Maintenant, en tant que Yahoo Store, ce logiciel est le créateur de boutique en ligne le plus populaire, avec plus de 20 000 utilisateurs.

Lorsque nous avons lancé Viaweb, personne ou presque ne comprenait ce que nous voulions dire lorsque nous disions que le logiciel fonctionnait sur le serveur.. Ce n'est que lorsque Hotmail a été lancé un an plus tard que les gens ont commencé à comprendre. Maintenant, tout le monde sait que c'est une approche valable. Il y a maintenant un nom pour ce que nous étions : un fournisseur de services d'application, ou ASP.

Je pense qu'une grande partie de la prochaine génération de logiciels sera écrite sur ce modèle. Même Microsoft, qui a le plus à perdre, semble voir l'inévitableté de déplacer certaines choses du bureau. Si le logiciel passe du bureau aux serveurs, cela signifiera un monde très différent pour les développeurs. Cet essai décrit les choses surprenantes que nous avons vues, comme certains des premiers visiteurs de ce nouveau monde. À la mesure dans laquelle le logiciel passe sur les serveurs, ce que je décris ici est l'avenir.

La prochaine chose ?

Lorsque nous revenons sur l'ère des logiciels de bureau, je pense que nous allons nous émerveiller des inconvénients que les gens en ont à supporter, tout comme nous nous émerveillons maintenant de ce que les premiers propriétaires de voitures ont à supporter. Pendant les vingt ou trente premières années, il a dû être un expert automobile pour posséder une voiture. Mais les voitures ont été une si grande victoire que beaucoup de gens qui n'étaient pas des experts en voitures voulaient les avoir aussi.

Les ordinateurs sont maintenant dans cette phase. Lorsque vous possédez un ordinateur de bureau, vous finissez par apprendre beaucoup plus que ce que vous vouliez savoir sur ce qui se passe à l'intérieur. Mais plus de la moitié des ménages aux États-Unis en possèdent un. Ma mère a un ordinateur qu'elle utilise pour le courrier électronique et pour tenir des comptes. Il y a quelques années, elle a été alarmée d'avoir reçu une lettre d'Apple, lui offrant un décompte sur une nouvelle version du système d'exploitation. Il y a quelque chose qui ne va pas quand une femme de soixante-cinq ans qui veut utiliser un ordinateur pour le courrier électronique et les comptes doit penser à installer de nouveaux systèmes d'exploitation. Les utilisateurs ordinaires ne devraient même pas connaître les mots "système d'exploitation", et encore moins "pilote d'appareil" ou "patch".

Il existe maintenant un autre moyen de fournir des logiciels qui éviteront aux utilisateurs de devenir administrateurs système. Les applications Web sont des programmes qui s'exécutent sur des serveurs Web et utilisent les pages Web comme interface utilisateur. Pour l'utilisateur moyen, ce nouveau type de logiciel sera plus facile, moins cher, plus mobile, plus fiable et souvent plus puissant que les logiciels de bureau.

Avec les logiciels basés sur le Web, la plupart des utilisateurs n'auront pas à penser à quoi que ce soit, sauf aux applications qu'ils utilisent. Toutes les choses désordonnées et changeantes seront assises sur un serveur quelque part, entretenues par le genre de personnes qui sont bonnes dans ce genre de choses. Et donc, vous n'aurez généralement pas besoin d'un ordinateur, en soi, pour utiliser un logiciel. Tout ce dont vous aurez besoin sera quelque chose avec un

clavier, un écran et un navigateur Web. Peut-être aura-t-il un accès Internet sans fil. Peut-être que ce sera aussi votre téléphone portable. Quoi qu'il en soit, ce sera de l'électronique grand public : quelque chose qui coûte environ 200 \$, et que les gens choisissent principalement en fonction de l'apparence de l'affaire. Vous paieriez plus pour les services Internet que pour le matériel, tout comme vous le faites maintenant avec les téléphones [1].

Il faudra environ un dixième de seconde pour un clic pour accéder au serveur et revenir, de sorte que les utilisateurs de logiciels fortement interactifs, comme Photoshop, voudront toujours que les calculs se déroulent sur le bureau. Mais si vous regardez le genre de choses pour lesquelles la plupart des gens utilisent les ordinateurs, une latence d'un dixième de seconde ne serait pas un problème. Ma mère n'a pas vraiment besoin d'un ordinateur de bureau, et il y a beaucoup de gens comme elle.

La victoire pour les utilisateurs

Près de chez moi, il y a une voiture avec un autocollant de pare-chocs qui indique « la mort avant le désagrément ». La plupart des gens, la plupart du temps, prendront n'importe quel choix qui nécessite le moins de travail. Si le logiciel basé sur le Web gagne, ce sera parce que c'est plus pratique. Et il semble que ce sera le cas, pour les utilisateurs et les développeurs à la fois.

Pour utiliser une application purement basée sur le Web, tout ce dont vous avez besoin est un navigateur connecté à Internet. Vous pouvez donc utiliser une application Web n'importe où. Lorsque vous installez un logiciel sur votre ordinateur de bureau, vous ne pouvez l'utiliser que sur cet ordinateur. Pire encore, vos fichiers sont piégés sur cet ordinateur. L'inconvénient de ce modèle devient de plus en plus évident à mesure que les gens s'habituent aux réseaux.

L'extrémité amincie du coin ici était le courrier électronique basé sur le Web. Des millions de gens se rendent maintenant compte que vous devriez avoir accès aux messages électroniques, où que vous soyez. Et si vous pouvez voir votre e-mail, pourquoi pas votre calendrier ? Si vous pouvez discuter d'un document avec vos collègues, pourquoi ne pouvez-vous pas le modifier ? Pourquoi l'une de vos données devrait-elle être piégée sur un ordinateur assis sur un bureau éloigné ?

Toute l'idée de "votre ordinateur" disparaît et est remplacée par "vos données". Vous devriez être en mesure d'obtenir vos données à partir de n'importe quel ordinateur. Ou plutôt, n'importe quel client, et un client n'a pas besoin d'être un ordinateur.

Les clients ne devraient pas stocker de données ; ils devraient être comme des téléphones. En fait, ils peuvent devenir des téléphones, ou vice versa. Et en tant que clients de plus en plus petit, vous avez une autre raison de ne pas garder vos données dessus : quelque chose que vous transportez avec vous peut être perdu ou volé. Laisser votre PDA dans un taxi est comme un accident de disque dur, sauf que vos données sont remises à quelqu'un d'autre au lieu d'être évaporées.

Avec un logiciel purement basé sur le Web, ni vos données ni les applications ne sont conservées sur le client. Vous n'avez donc pas besoin d'installer quoi que ce soit pour l'utiliser. Et lorsqu'il n'y a pas d'installation, vous n'avez pas à vous inquiéter d'un mauvais problème d'installation. Il ne peut pas y avoir d'incompatibilité entre l'application et votre système d'exploitation, car le logiciel ne fonctionne pas sur votre système d'exploitation.

Parce qu'il n'a pas besoin d'installation, il sera facile et courant d'essayer un logiciel Web avant de l'acheter. Vous devriez vous attendre à pouvoir tester n'importe quelle application Web gratuitement, simplement en vous rendant sur le site où elle est proposée. Chez Viaweb, l'ensemble de notre site était comme une grosse flèche pointant les utilisateurs vers l'essai routier.

Après avoir essayé la démo, l'inscription au service ne devrait nécessiter rien de plus que de remplir un bref formulaire. Et cela devrait être le dernier travail que l'utilisateur doit faire. Avec les logiciels basés sur le Web, vous devriez obtenir de nouvelles versions sans payer de supplément, ni faire de travail, ni peut-être même le savoir.

Les mises à jour ne seront pas les grands chocs qu'elles sont maintenant. Au fil du temps, les applications deviendront progressivement de plus en plus puissantes. Cela demandera un certain effort de la part des développeurs. Ils devront concevoir le logiciel afin qu'il puisse être mis à jour sans confondre les utilisateurs. C'est un nouveau problème, mais il y a des moyens de le résoudre.

Avec les applications Web, tout le monde utilise la même version, et les bugs peuvent être corrigés dès qu'ils sont découverts. Les logiciels Web devraient donc avoir beaucoup moins de bugs que les logiciels de bureau. Chez Viaweb, je doute que nous ayons jamais eu dix bugs connus à un moment donné. C'est des ordres de grandeur meilleurs que les logiciels de bureau.

Les applications Web peuvent être utilisées par plusieurs personnes en même temps. C'est une victoire évidente pour les applications collaboratives, mais je parie que les utilisateurs commenceront à le vouloir dans la plupart des applications une fois qu'ils se rendront compte que c'est possible. Il sera souvent utile de laisser deux personnes modifier le même document, par exemple. Viaweb a permis à plusieurs utilisateurs de modifier un site simultanément, d'autant plus que c'était la bonne façon d'écrire le logiciel que parce que nous nous attendions à ce que les utilisateurs le veuillent, mais il s'est avéré que beaucoup l'ont fait.

Lorsque vous utilisez une application Web, vos données seront plus en sécurité. Les plantages de disque ne seront pas une chose du passé, mais les utilisateurs n'en entendront plus parler. Ils se produiront au sein des fermes de serveurs. Et les entreprises offrant des applications Web feront en fait des sauvegardes - non seulement parce qu'elles auront de vrais administrateurs système qui s'inquiètent de telles choses, mais aussi parce qu'un ASP qui perd les données des gens aura de gros problèmes. Lorsque les gens perdent leurs propres données lors d'un plantage de disque, ils ne peuvent pas se mettre en colère, parce qu'ils n'ont qu'eux-mêmes à être en colère. Lorsqu'une entreprise perd leurs données pour eux, ils seront beaucoup plus furieux.

Enfin, les logiciels basés sur le Web devraient être moins vulnérables aux virus. Si le client n'exécute rien d'autre qu'un navigateur, il y a moins de chances d'exécuter des virus et aucune donnée localement à endommager. Et un programme qui a attaqué les serveurs eux-mêmes devrait les trouver bien défendus [2].

Pour les utilisateurs, les logiciels basés sur le web seront moins stressants. Je pense que chez l'utilisateur moyen de Windows, vous trouveriez un énorme et pratiquement inexploité désir pour des logiciels répondant à cette description. Libéré, il pourrait s'agir d'une force puissante.

La Cité du Code

Pour les développeurs, la différence la plus notable entre les logiciels Web et les logiciels de bureau est qu'une application Web n'est pas un seul morceau de code. Il s'agit d'une collection de programmes de différents types plutôt que d'un seul grand binaire. Et donc, concevoir des logiciels basés sur le Web, c'est comme concevoir une ville plutôt qu'un bâtiment : ainsi que des bâtiments, vous avez besoin de routes, de panneaux de signalisation, de services publics, de police et de services d'incendie, et de plans pour la croissance et divers types de catastrophes.

Chez Viaweb, les logiciels incluent des applications assez importantes auxquelles les utilisateurs parlaient directement, des programmes
ccccbcdhkglhfluddlnkbhrcdvncrcbkbgflefjbvi
que d'autres programmes utilisaient, des programmes qui s'exerçaient constamment en arrière-plan à la recherche de problèmes, des programmes qui essayaient de démarrer les choses en cas de panne, des programmes qui s'exécutaient occasionnellement pour compiler des statistiques ou construire des index pour les recherches, des programmes que nous avons couru explicitement vers des ressources de collecte de déchets ou pour déplacer ou restaurer des données, des programmes qui prétendaient être des utilisateurs (pour mesurer les performances ou exposer des bugs), des programmes pour diagnostiquer les problèmes de réseau, des programmes pour faire des sauvegardes, des interfaces avec des services. Trevor Blackwell a écrit un programme spectaculaire pour déplacer les magasins vers de nouveaux serveurs à travers le pays, sans les fermer, après que nous ayons été achetés par Yahoo. Les programmes nous ont téléphonés, ont envoyé des fax et des courriels aux utilisateurs, ont effectué des transactions avec des fournisseurs de cartes de crédit et se sont entretenus par le biais de sockets, de tuyaux, de requêtes HTTP, de SSH, de paquets UDP, de mémoire partagée et de fichiers. Certains de Viaweb consistaient même en l'absence de programmes, puisque l'une des clés de la sécurité Unix est de ne pas exécuter des utilitaires inutiles que les gens pourraient utiliser pour pénétrer dans vos serveurs.

Cela ne s'est pas terminé avec le logiciel. Nous avons passé beaucoup de temps à réfléchir aux configurations des serveurs. Nous avons construit les serveurs nous-mêmes, à partir de composants, en partie pour économiser de l'argent, et

en partie pour obtenir exactement ce que nous voulions. Nous avons dû nous demander si notre FAI en amont avait des connexions assez rapides à toutes les dorsales. Nous avons daté en série les fournisseurs de RAID.

Mais le matériel n'est pas seulement quelque chose dont il y a à craindre. Lorsque vous le contrôlez, vous pouvez en faire plus pour les utilisateurs. Avec une application de bureau, vous pouvez spécifier un certain matériel minimum, mais vous ne pouvez pas en ajouter d'autres. Si vous administrez les serveurs, vous pouvez en une seule étape permettre à tous vos utilisateurs de contacter des personnes, ou d'envoyer des fax, ou d'envoyer des commandes par téléphone, ou de traiter des cartes de crédit, etc., simplement en installant le matériel approprié. Nous avons toujours cherché de nouvelles façons d'ajouter des fonctionnalités avec du matériel, non seulement parce que cela plaisait aux utilisateurs, mais aussi comme un moyen de nous distinguer des concurrents qui (soit parce qu'ils vendaient des logiciels de bureau, soit parce qu'ils revendaient des applications Web par l'intermédiaire de FAI) n'avaient pas de contrôle direct sur le matériel.

Parce que le logiciel d'une application Web sera une collection de programmes plutôt qu'un seul binaire, il peut être écrit dans n'importe quel nombre de langues différentes. Lorsque vous écrivez un logiciel haut de gamme, vous êtes pratiquement obligé d'écrire l'application dans le même langage que le système d'exploitation sous-jacent, c'est-à-dire C et C++. Et donc ces langages (en particulier chez les personnes non technologiques comme les gestionnaires et les investisseurs de capital-risque) ont dû être considérés comme les langages pour le développement de logiciels « sérieux ». Mais ce n'était qu'un artefact de la façon dont les logiciels de bureau devaient être livrés. Pour les logiciels basés sur serveur, vous pouvez utiliser n'importe quel langage que vous voulez [3]. Aujourd'hui, beaucoup des meilleurs pirates utilisent des langages loin de C et C++ : Perl, Python et même Lisp.

Avec les logiciels sur serveur, personne ne peut vous dire quelle langue utiliser, car vous contrôlez l'ensemble du système, jusqu'au matériel. Différentes langues sont bonnes pour différentes tâches. Vous pouvez utiliser ce qui est le mieux pour chacune. Et lorsque vous avez des concurrents, "vous pouvez" signifie "vous devez" (nous y reviendrons plus tard), car si vous ne profitez pas de cette possibilité, vos concurrents le feront.

La plupart de nos concurrents utilisaient C et C++, ce qui rendait leur logiciel visiblement inférieur parce que (entre autres choses), ils n'avaient aucun moyen de contourner l'apatriodie des scripts CGI. Si vous deviez changer quelque chose, tous les changements devaient se produire sur une seule page, avec un bouton de mise à jour en bas. Comme je l'explique dans le chapitre 12, en utilisant Lisp, que beaucoup de gens considèrent encore comme un langage de recherche, nous pourrions faire en sorte que l'éditeur Viaweb se comporte davantage comme un logiciel de bureau.

Sorties

L'un des changements les plus importants dans ce nouveau monde est la façon dont vous faites les sorties. Dans le secteur des logiciels de bureau, faire une version est un énorme traumatisme, dans lequel toute l'entreprise transpire et s'efforce de pousser un seul morceau de code géant. Des comparaisons évidentes se suggèrent, à la fois pour le processus et pour le produit qui en résulte.

Avec un logiciel sur serveur, vous pouvez apporter des modifications presque comme vous le feriez dans un programme que vous écriviez pour vous-même. Vous publiez un logiciel sous la forme d'une série de changements incrémentiels au lieu d'une grande explosion occasionnelle. Une société de logiciels de bureau typique pourrait faire une ou deux versions par an. Chez Viaweb, nous faisions souvent trois à cinq sorties par jour.

Lorsque vous passez à ce nouveau modèle, vous vous rendez compte à quel point le développement logiciel est affecté par la façon dont il est publié. Bon nombre des problèmes les plus désagréables que vous voyez dans le secteur des logiciels de bureau sont dus à la nature catastrophique des versions.

Lorsque vous ne publiez qu'une seule nouvelle version par an, vous avez tendance à faire face à des bugs en gros. Quelque temps avant la date de sortie, vous assemblez une nouvelle version dans laquelle la moitié du code a été arrachée et remplacée, introduisant d'innombrables bugs. Ensuite, une équipe de personnes chargées de l'assurance qualité intervient et commence à les compter, et les programmeurs travaillent sur la liste, les corrigent. Ils n'amènent généralement pas à la fin de la liste, et en effet, personne ne sait où se trouve la

fin. C'est comme pêcher des décombres dans un étang. On ne sait jamais vraiment ce qui se passe à l'intérieur du logiciel. Au mieux, vous vous retrouvez avec une sorte d'exactitude statistique.

Avec les logiciels basés sur serveur, la majeure partie du changement est faible et incrémentielle. Cela en soi est moins susceptible d'introduire des bugs. Cela signifie également que vous savez quoi tester le plus soigneusement lorsque vous êtes sur le point de publier un logiciel : la dernière chose que vous avez changée. Vous vous retrouvez avec une emprise beaucoup plus ferme sur le code. En règle générale, vous savez ce qui se passe à l'intérieur. Vous n'avez pas le code source mémorisé, bien sûr, mais lorsque vous lisez la source, vous le faites comme un pilote scannant le tableau de bord, pas comme un détective essayant de résoudre un mystère.

Les logiciels de bureau engendrent un certain fatalisme à propos des bugs. Vous savez que vous expédiez quelque chose chargé de bugs, et vous avez même mis en place des mécanismes pour le compenser (par exemple, les versions de correctifs). Alors pourquoi s'inquiéter d'un peu plus ? Bientôt, vous publiez des fonctionnalités complètes dont vous savez qu'elles sont cassées. Apple l'a fait il y a quelques années. Ils se sont sentis sous pression pour sortir leur nouveau système d'exploitation, dont la date de sortie avait déjà glissé quatre fois, mais une partie du logiciel (prise en charge de CD et DVD) n'était pas prête. La solution ? Ils ont sorti le système d'exploitation sans les pièces inachevées, et les utilisateurs ont dû l'installer plus tard.

Avec les logiciels Web, vous n'avez jamais à publier de logiciel avant qu'il ne fonctionne, et vous pouvez le libérer dès qu'il fonctionne.

Le vétéran de l'industrie pense peut-être : c'est une bonne idée de dire que vous n'avez jamais à publier de logiciel avant qu'il ne fonctionne, mais que se passe-t-il lorsque vous avez promis de livrer une nouvelle version de votre logiciel à une certaine date ? Avec un logiciel basé sur le Web, vous ne feriez pas une telle promesse, car il n'y a pas de versions. Votre logiciel change progressivement et continuellement. Certains changements peuvent être plus importants que d'autres, mais l'idée de versions ne s'adapte naturellement aux logiciels basés sur le Web.

Si quelqu'un se souvient de Viaweb, cela peut sembler bizarre, parce que nous annoncions toujours de nouvelles versions. Cela a été fait entièrement à des fins de relations publiques. La presse professionnelle, nous l'avons appris, pense en chiffres de version. Ils vous donneront une couverture majeure pour une version majeure, c'est-à-dire un nouveau premier chiffre sur le numéro de version, et généralement un paragraphe tout au plus pour une version ponctuelle, c'est-à-dire un nouveau chiffre après le point décimal.

Certains de nos concurrents proposaient des logiciels de bureau et avaient en fait des numéros de version. Et pour ces communiqués, dont le simple fait nous semblait la preuve de leur retard, ils obtiendraient toutes sortes de publicités. Nous ne voulions pas manquer, alors nous avons commencé à donner des numéros de version à notre logiciel aussi. Lorsque nous voulions de la publicité, nous faisions une liste de toutes les fonctionnalités que nous avions ajoutées depuis la dernière "version", en collant un nouveau numéro de version sur le logiciel et en publiant un communiqué de presse disant que la nouvelle version était disponible immédiatement. Étonnamment, personne ne nous a jamais appelé.

Au moment où nous avons été achetés, nous l'avions fait trois fois, nous étions donc sur la version 4. Version 4.1 si je me souviens bien. Une fois que Viaweb est devenu Yahoo Store, il n'y avait plus un besoin aussi désespéré de publicité, donc bien que le logiciel ait continué à évoluer, toute l'idée des numéros de version a été tranquillement abandonnée.

Bugs

L'autre avantage technique majeur des logiciels basés sur le Web est que vous pouvez reproduire la plupart des bugs. Vous avez les données des utilisateurs sur votre disque. Si quelqu'un casse votre logiciel, vous n'avez pas besoin d'essayer de deviner ce qui se passe, comme vous le feriez avec un logiciel de bureau : vous devriez être en mesure de reproduire l'erreur pendant qu'il est au téléphone avec vous. Vous le savez peut-être déjà, si vous avez un code pour remarquer les erreurs intégrées à votre application.

Les logiciels Web sont utilisés 24 heures sur 24, de sorte que tout ce que vous faites est immédiatement passé par l'essoreuse. Les bugs se lèvent rapidement.

Les sociétés de logiciels sont parfois accusées de laisser les utilisateurs débuguer leurs logiciels. Et c'est exactement ce que je préconise. Pour les logiciels basés sur le Web, c'est en fait un bon plan, car les bugs sont moins nombreux et transitoires. Lorsque vous publiez progressivement un logiciel, vous obtenez beaucoup moins de bugs pour commencer. Et lorsque vous pouvez reproduire les erreurs et publier des modifications instantanément, vous pouvez trouver et corriger la plupart des bugs dès qu'ils apparaissent. Nous n'avons jamais eu assez de bugs à un moment donné pour nous soucier d'un système formel de suivi des bugs.

Vous devriez tester les modifications avant de les publier, bien sûr, de sorte qu'aucun bug majeur ne soit publié. Les quelques personnes qui s'échappent inévitablement impliqueront des cas limites et n'affecteront que les quelques utilisateurs qui les rencontrent avant que quelqu'un n'appelle pour se plaindre. Tant que vous corrigez les bugs tout de suite, l'effet net, pour l'utilisateur moyen, est de beaucoup moins de bugs. Je doute que l'utilisateur moyen de Viaweb ait jamais vu un bug.

Il est plus facile de réparer de nouveaux bugs que d'anciens. Il est généralement assez rapide de trouver un bug dans le code que vous venez d'écrire. Quand il s'avère, vous savez souvent ce qui ne va pas avant même de regarder la source, parce que vous vous en inquiétez déjà inconsciemment. Corriger un bug dans quelque chose que vous avez écrit il y a six mois (le cas moyen si vous publiez une fois par an) est beaucoup plus de travail. Et comme vous ne comprenez pas non plus le code, vous êtes plus susceptible de le corriger d'une manière laide, ou même d'introduire plus de bugs [4].

Lorsque vous attrapez des bugs tôt, vous obtenez également moins de bugs composés. Les bugs composés sont deux bugs séparés qui interagissent : vous descendez en bas, et lorsque vous atteignez la main courante, elle se détache dans votre main. Dans les logiciels, ce type de bug est le plus difficile à trouver, et a également tendance à avoir les pires conséquences [5]. L'approche

traditionnelle "tout casser, puis filtrer les bugs" donne intrinsèquement beaucoup de bugs composés. Et les logiciels publiés dans une série de petits changements ont intrinsèquement tendance à ne pas le faire. Les sols sont constamment balayés pour enlever tout objet qui pourrait se coincer dans quelque chose.

Cela aide si vous utilisez une technique appelée "programmation fonctionnelle". La programmation fonctionnelle signifie éviter les effets secondaires. C'est quelque chose que vous êtes plus susceptible de voir dans les documents de recherche que dans les logiciels commerciaux, mais pour les applications Web, cela s'avère vraiment utile. Il est difficile d'écrire des programmes entiers en tant que code purement fonctionnel, mais vous pouvez écrire des morceaux substantiels de cette façon. Cela rend ces parties de votre logiciel plus faciles à tester, car elles n'ont pas d'état, et c'est très pratique dans une situation où vous faites et testez constamment de petites modifications. J'ai écrit une grande partie de l'éditeur de Viaweb dans ce style, et nous avons fait de notre langage de script, RTML, un langage purement fonctionnel.

Les gens du secteur des logiciels de bureau trouveront cela difficile à créditer, mais chez Viaweb, les bugs sont devenus presque un jeu. Étant donné que la plupart des bugs publiés concernaient des cas limites, les utilisateurs qui les rencontraient étaient susceptibles d'être des utilisateurs avancés, repoussant les limites. Les utilisateurs avertis par la publicité sont plus indulgents à propos des bugs, d'autant plus que vous les avez probablement introduits au cours de l'ajout d'une fonctionnalité qu'ils demandaient. En fait, parce que les bugs étaient rares et que vous deviez faire des choses sophistiquées pour les voir, les utilisateurs avancés étaient souvent fiers d'en attraper un. Ils appelleraient le soutien dans un esprit plus de triomphe que de colère, comme s'ils nous avaient marqué des points.

Support

Lorsque vous pouvez reproduire des erreurs, cela change votre approche du support client. Dans la plupart des entreprises de logiciels, une assistance est offerte comme un moyen de faire en sorte que les clients se sentent mieux. Soit ils vous appellent à propos d'un bug connu, soit ils font juste quelque chose de mal et vous devez comprendre quoi. Dans les deux cas, il n'y a pas grand-chose

que vous puissiez apprendre d'eux. Vous avez donc tendance à considérer les appels à l'assistance comme une "douleur dans le cul (*pain in the ass*)" que vous voulez isoler de vos développeurs autant que possible.

Ce n'était pas comme ça que les choses fonctionnaient chez Viaweb. Chez Viaweb, l'assistance était gratuite, parce que nous voulions avoir des nouvelles des clients. Si quelqu'un avait un problème, nous voulions le savoir tout de suite afin de pouvoir reproduire l'erreur et publier un correctif.

Ainsi, chez Viaweb, les développeurs étaient toujours en contact étroit avec le support. Les gens du support client étaient à environ 10 mètres des programmeurs, et savaient qu'ils pouvaient toujours interrompre n'importe quoi avec un rapport d'un véritable bug. Nous quittons une réunion du conseil d'administration pour corriger un grave bug.

Notre approche du soutien a rendu tout le monde plus heureux. Les clients étaient ravis. Imaginez ce que cela ferait d'appeler une ligne d'assistance et d'être traité comme quelqu'un qui apporte des nouvelles importantes. Les gens du support client l'ont aimé parce que cela signifiait qu'ils pouvaient aider les utilisateurs, au lieu de leur lire des scripts. Et les professionnels l'ont aimé parce qu'ils pouvaient reproduire des bugs au lieu de simplement entendre de vagues rapports de seconde main à leur sujet.

Notre politique de correction des bugs à la volée a changé la relation entre les personnes du support client et les hackers. Dans la plupart des entreprises de logiciels, les personnes de soutien sont des boucliers humains sous-payés, et les hackers sont de petites copies de Dieu le Père, créateurs du monde. Quelle que soit la procédure de signalement des bogues, il est probable qu'elle soit à sens unique : aider les personnes qui entendent parler de bugs en remplissant un formulaire qui est finalement transmis (éventuellement via l'assurance qualité) aux programmeurs, qui le mettent sur leur liste de choses à faire. C'était différent chez Viaweb. Dans la minute de l'annonce d'un bug de la part d'un client, les personnes du support pouvaient se tenir à côté d'un programmeur et l'entendre dire "Merde, tu as raison, c'est un bug". Cela a ravi les gens de soutien d'entendre que "vous avez raison" de la part des hackers. Ils nous apportaient des bugs avec le même air expectant qu'un chat qui vous apportait une souris qu'il vient de tuer. Cela les a également rendus plus

prudents dans le jugement de la gravité d'un bug, parce que maintenant leur honneur était en jeu.

Après que nous ayons été achetés par Yahoo, les gens du support client ont été éloignés des programmeurs. Ce n'est qu'à ce moment-là que nous avons réalisé qu'il s'agissait effectivement d'une assurance qualité et, dans une certaine mesure, d'un certain marketing. En plus d'attraper des bugs, ils étaient les gardiens de la connaissance de choses plus vagues et de type bug, comme des fonctionnalités qui confondaient les utilisateurs [6]. Ils constituaient également une sorte de groupe de discussion par procuration ; nous pouvions leur demander laquelle de deux nouvelles fonctionnalités les utilisateurs souhaitaient le plus, et ils avaient toujours raison.

Morale

Être capable de publier un logiciel immédiatement est une grande source de motivation. Souvent, alors que je me rendais au travail à pied, je pensais à un changement que je voulais apporter au logiciel, et je le faisais ce jour-là. Cela a également fonctionné pour de plus grandes fonctionnalités. Même si quelque chose allait prendre deux semaines à écrire (les projets prenaient plus de temps), je savais que je pouvais voir l'effet dans le logiciel dès qu'il était terminé.

Si j'avais dû attendre un an pour la prochaine sortie, j'aurais mis de côté la plupart de ces idées, au moins pendant un certain temps. La chose à propos des idées, cependant, c'est qu'elles mènent à plus d'idées. Avez-vous déjà remarqué que lorsque vous vous asseyez pour écrire quelque chose, la moitié des idées qui s'y terminent sont celles auxquelles vous avez pensé en écrivant ? Il en va de même pour les logiciels. Travailler à la mise en œuvre d'une idée vous donne plus d'idées. La mise en suspens d'une idée vous coûte donc non seulement ce retard dans sa mise en œuvre, mais aussi toutes les idées que sa mise en œuvre aurait permis. En fait, la mise en suspens d'une idée inhibe probablement même les nouvelles idées : lorsque vous commencez à penser à une nouvelle fonctionnalité, vous apercevez l'étagère et vous vous dites "mais j'ai déjà beaucoup de nouvelles choses que je veux pour la prochaine version".

Ce que font les grandes entreprises au lieu de mettre en œuvre des fonctionnalités, c'est de les planifier. Chez Viaweb, nous avons parfois eu des problèmes sur ce compte. Les investisseurs et les analystes nous demandaient ce que nous avions prévu pour l'avenir. La réponse vérifique aurait été que nous n'avions aucun plan. Nous avions des idées générales sur les choses que nous voulions améliorer, mais si nous savions comment nous l'aurions déjà fait. Qu'allions-nous faire au cours des six prochains mois ? Tout cela semblait être la plus grande victoire. Je ne sais pas si j'ai jamais osé donner cette réponse, mais c'était la vérité. Les plans ne sont qu'un autre mot pour les idées sur l'étagère. Lorsque nous avons pensé à de bonnes idées, nous les avons mises en œuvre.

Chez Viaweb, comme dans de nombreuses sociétés de logiciels, la plupart des codes avaient un propriétaire défini. Mais lorsque vous possédiez quelque chose, vous le possédiez vraiment : personne, à l'exception du propriétaire d'un logiciel, n'avait à approuver (ou même à connaître) une version. Il n'y avait pas de protection contre la casse, sauf la peur de ressembler à un idiot à ses pairs, et c'était plus que suffisant. J'ai peut-être donné l'impression que nous venions d'avancer l'écriture de code. Nous sommes allés vite, mais nous avons réfléchi très attentivement avant de publier des logiciels sur ces serveurs. Et faire attention est plus important pour la fiabilité que d'avancer lentement. Parce qu'il fait très attention, un pilote de la Marine peut faire atterrir un avion de 40 000 lb à 140 miles à l'heure sur un pont de porte-avions, la nuit, plus en toute sécurité que l'adolescent moyen ne peut couper un bagel.

Cette façon d'écrire des logiciels est une épée à double tranchant, bien sûr. Cela fonctionne beaucoup mieux pour une petite équipe de bons programmeurs de confiance que pour une grande entreprise de programmeurs médiocres, où les mauvaises idées sont prises par les comités au lieu des personnes qui les ont eues.

Le Ruisseau à l'envers

Heureusement, les logiciels basés sur le Web nécessitent moins de programmeurs. J'ai déjà travaillé pour une société de logiciels de bureau de taille moyenne qui comptait plus de 100 personnes travaillant dans l'ingénierie dans son ensemble. Seuls 13 d'entre eux étaient en cours de développement de

produits. Tous les autres travaillaient sur les versions, les ports, et ainsi de suite. Avec les logiciels basés sur le Web, tout ce dont vous avez besoin (au plus) sont les 13 personnes, car il n'y a pas de versions, de ports, et ainsi de suite.

Viaweb a été écrit par seulement trois personnes [7]. J'étais toujours sous pression pour embaucher plus, parce que nous voulions nous faire acheter, et nous savions que les acheteurs auraient du mal à payer un prix élevé pour une entreprise avec seulement trois programmeurs. (Solution : nous en avons embauché plus, mais nous avons créé de nouveaux projets pour eux.)

Lorsque vous pouvez écrire des logiciels avec moins de programmeurs, cela vous permet d'économiser plus que de l'argent. Comme Fred Brooks l'a souligné dans *The Mythical Man-Month*, l'ajout de personnes à un projet a tendance à le ralentir. Le nombre de connexions possibles entre les développeurs augmente de façon exponentielle avec la taille du groupe [8]. Plus le groupe est grand, plus ils passeront de temps dans des réunions à négocier la façon dont leur logiciel fonctionnera ensemble, et plus ils obtiendront de bugs à partir d'interactions imprévues. Heureusement, ce processus fonctionne également à l'envers : à mesure que les groupes deviennent plus petits, le développement de logiciels devient exponentiellement plus efficace. Je ne me souviens pas que les programmeurs de Viaweb aient jamais eu une véritable réunion. Nous n'avons jamais eu plus à dire à un moment donné que ce que nous pouvions dire alors que nous allions déjeuner.

S'il y a un inconvénient ici, c'est que tous les programmeurs doivent également être, dans une certaine mesure, des administrateurs système. Lorsque vous hébergez un logiciel, quelqu'un doit surveiller les serveurs, et dans la pratique, les seules personnes qui peuvent le faire correctement sont celles qui ont écrit le logiciel. Chez Viaweb, notre système avait tellement de composants et changeait si fréquemment qu'il n'y avait pas de frontière précise entre le logiciel et l'infrastructure. La déclaration arbitraire d'une telle frontière aurait limité nos choix de conception. Et donc, même si nous espérions constamment qu'un jour ("dans quelques mois") tout serait suffisamment stable pour que nous puissions embaucher quelqu'un dont le travail était juste de s'inquiéter des serveurs, cela ne s'est jamais produit.

Je ne pense pas que cela puisse être autrement, tant que vous développez toujours activement le produit. Les logiciels basés sur le Web ne sont jamais quelque chose que vous écrivez, que vous enregistrez et que vous rentrez chez vous. C'est une chose en direct, qui fonctionne sur vos serveurs en ce moment. Un mauvais bug ne pourrait pas simplement faire planter le processus d'un utilisateur ; il pourrait tous les faire planter. Si un bug dans votre code corrompt certaines données sur le disque, vous devez le corriger. Et ainsi de suite. Nous avons constaté que vous n'avez pas besoin de regarder les serveurs toutes les minutes (après la première année environ), mais vous voulez certainement garder un œil sur les choses que vous avez changées récemment. Vous ne relâchez pas le code tard dans la nuit, puis vous rentrez chez vous.

Observer les utilisateurs

Avec les logiciels sur serveur, vous êtes en contact étroit avec votre code. Vous pouvez également être en contact plus étroit avec vos utilisateurs. Intuit est célèbre pour se présenter aux clients dans les magasins de détail et leur demander de les suivre à la maison. Si vous avez déjà vu quelqu'un utiliser votre logiciel pour la première fois, vous savez quelles surprises l'attendaient.

Les logiciels devraient faire ce que les utilisateurs pensent qu'il fera. Mais vous ne pouvez avoir aucune idée de ce que les utilisateurs penseront, croyez-moi, jusqu'à ce que vous les regardiez. Et les logiciels basés sur serveur vous donnent des informations sans précédent sur leur comportement. Vous n'êtes pas limité à de petits groupes de discussion artificiels. Vous pouvez voir chaque clic effectué par chaque utilisateur. Vous devez examiner attentivement ce que vous allez examiner, car vous ne voulez pas violer la vie privée des utilisateurs, mais même l'échantillonnage statistique le plus général peut être très utile.

Lorsque vous avez les utilisateurs sur votre serveur, vous n'avez pas à vous fier aux points de repère, par exemple. Les points de repère sont des utilisateurs simulés. Avec un logiciel basé sur serveur, vous pouvez regarder les utilisateurs réels. Pour décider ce qu'il faut optimiser, il suffit de se connecter à un serveur et de voir ce qui consomme tout le CPU. Et vous savez quand arrêter d'optimiser aussi : nous avons finalement amené l'éditeur Viaweb au point où il était lié à la mémoire plutôt qu'au CPU, et comme nous ne pouvions rien faire

pour réduire la taille des données des utilisateurs (enfin, rien de facile), nous savions que nous pouvions aussi bien nous arrêter là.

L'efficacité est importante pour les logiciels basés sur serveur, car vous payez pour le matériel. Le nombre d'utilisateurs que vous pouvez prendre en charge par serveur est le diviseur de votre coût en capital, donc si vous pouvez rendre votre logiciel très efficace, vous pouvez sous-estimer vos concurrents tout en faisant un profit. Chez Viaweb, nous avons réduit le coût en capital par utilisateur à environ 5 \$. Ce serait moins maintenant, probablement moins que le coût de leur envoyer la facture du premier mois. Le matériel est gratuit maintenant, si votre logiciel est raisonnablement efficace.

Observer les utilisateurs peut vous guider dans la conception ainsi que dans l'optimisation. Viaweb avait un langage de script appelé RTML qui permettait aux utilisateurs avancés de définir leurs propres styles de page. Nous avons découvert que RTML est devenu une sorte de boîte de suggestion, parce que les utilisateurs ne l'utilisaient que lorsque les styles de page prédéfinis ne pouvaient pas faire ce qu'ils voulaient. À l'origine, l'éditeur a mis des barres de boutons sur la page, par exemple, mais après qu'un certain nombre d'utilisateurs ont utilisé RTML pour mettre des boutons sur le côté gauche, nous en avons fait la valeur par défaut dans les styles de page prédéfinis.

Enfin, en observant les utilisateurs, vous pouvez souvent savoir quand ils ont des ennuis. Et puisque le client a toujours raison, c'est un signe de quelque chose que vous devez réparer. Chez Viaweb, la clé pour obtenir des utilisateurs était le test en ligne. Il ne s'agissait pas seulement d'une série de diapositives construites par des gens du marketing. Lors de notre test, les utilisateurs ont effectivement utilisé le logiciel. Cela a pris environ cinq minutes, et à la fin, ils avaient construit un vrai magasin qui fonctionnait.

Le test a été la façon dont nous avons obtenu presque tous nos nouveaux utilisateurs. Je pense que ce sera la même chose pour la plupart des applications Web. Si les utilisateurs peuvent réussir un test, ils aimeront le produit. S'ils sont confus ou s'ennuient, ils ne le feront pas. Donc, tout ce que nous pourrions faire pour amener plus de gens à passer le test augmenterait notre taux de croissance.

J'ai étudié les pistes de clics des personnes qui font le test et j'ai constaté qu'à une certaine étape, elles se confondaient et cliquaient sur le bouton Retour du navigateur. (Si vous essayez d'écrire des applications Web, vous constaterez que le bouton Retour devient l'un de vos problèmes philosophiques les plus intéressants.) J'ai donc ajouté un message à ce moment-là, disant aux utilisateurs qu'ils avaient presque terminé et leur rappelant de ne pas cliquer sur le bouton Retour. Une autre bonne chose à propos des logiciels basés sur le Web est que vous obtenez une rétroaction instantanée des changements : le nombre de personnes qui terminent le test est immédiatement passé de 60 % à 90 %. Et comme le nombre de nouveaux utilisateurs était fonction du nombre d'essais terminés, notre croissance des revenus a augmenté de 50 %, juste à cause de ce changement.

L'Argent

Au début des années 1990, j'ai lu un article qui décrivait les logiciels comme une « entreprise d'abonnement ». Au début, cela semblait une déclaration très cynique. Mais plus tard, j'ai réalisé que cela reflète la réalité : le développement logiciel est un processus continu. Je pense que c'est plus propre si vous facturez ouvertement des frais d'abonnement, au lieu de forcer les gens à continuer à acheter et à installer de nouvelles versions pour qu'ils continuent à vous payer. Et pour l'accord, les abonnements sont le moyen naturel de facturer les applications Web.

L'hébergement des applications est un domaine où les entreprises joueront un rôle qui n'est pas susceptible d'être rempli par des logiciels gratuits. L'hébergement d'applications est très stressant et comporte des dépenses réelles. Personne ne voudra le faire gratuitement.

Pour les entreprises, les applications Web sont une source de revenus idéale. Au lieu de commencer chaque trimestre avec une ardoise vierge, vous avez une source de revenus récurrents. Parce que votre logiciel évolue progressivement, vous n'avez pas à vous inquiéter qu'un nouveau modèle échoue. Il n'y a jamais besoin d'un nouveau modèle, en soi, et si vous faites quelque chose au logiciel que les utilisateurs détestent, vous le saurez tout de suite. Vous n'avez aucun problème avec les factures irrécouvrables ; si quelqu'un

ne paie pas, vous pouvez simplement éteindre le service. Et il n'y a aucune possibilité de piratage.

Ce dernier « avantage » peut s'être révélé être un problème. Une certaine quantité de bidouillage est à l'avantage des entreprises de logiciels. Si un utilisateur n'avait jamais acheté votre logiciel à quelque prix que ce soit, vous n'avez rien perdu s'il utilise une copie hackée. En fait, vous gagnez, parce qu'il est un utilisateur de plus qui aide à faire de votre logiciel la norme - ou qui pourrait en acheter une copie plus tard, lorsqu'il obtiendra son diplôme d'études secondaires.

Lorsqu'elles le peuvent, les entreprises aiment faire ce qu'on appelle la discrimination des prix, ce qui signifie facturer à chaque client autant qu'elles le peuvent [9]. Les logiciels sont particulièrement adaptés à la discrimination des prix, car le coût marginal est proche de zéro. C'est pourquoi certains logiciels coûtent plus cher pour fonctionner sur Suns que sur les boîtiers Intel : une entreprise qui utilise Suns n'est pas intéressée à économiser de l'argent et peut être facturée plus cher en toute sécurité. Le bidouillage est en fait le plus bas niveau de discrimination par les prix. Je pense que les éditeurs de logiciels comprennent cela et ferment délibérément les yeux sur certains types de bidouillage [10]. Avec les logiciels basés sur le serveur, elles devront trouver une autre solution.

Les logiciels basés sur le Web se vendent bien, surtout par rapport aux logiciels de bureau, car il est facile à acheter. Vous pourriez penser que les gens décident d'acheter quelque chose, puis de l'acheter, en deux étapes distinctes. C'est ce que je pensais avant Viaweb, dans la mesure où j'ai pensé à la question. En fait, la deuxième étape peut se propager à la première : si quelque chose est difficile à acheter, les gens changeront d'avis sur la question de savoir s'ils le voulaient. Et vice versa : vous vendrez plus de quelque chose quand il sera facile à acheter. J'achète plus de nouveaux livres parce qu'Amazon existe. Les logiciels basés sur le Web sont à peu près la chose la plus facile au monde à acheter, surtout si vous venez de faire une démo en ligne. Les utilisateurs ne devraient pas avoir à faire beaucoup plus que d'entrer un numéro de carte de crédit. (Faites-leur en faire plus à vos périls.)

Parfois, les logiciels basés sur le Web sont proposés par les FAI agissant en tant que revendeurs. C'est une mauvaise idée. Vous devez administrer les serveurs, car vous devez constamment améliorer à la fois le matériel et les logiciels. Si vous renoncez au contrôle direct des serveurs, vous renoncez à la plupart des avantages du développement d'applications basées sur le Web.

Plusieurs de nos concurrents se sont tiré une balle dans le pied de cette façon - généralement, je pense, parce qu'ils étaient envahis par des costumes qui étaient enthousiasmés par cet énorme canal potentiel, et ne se rendaient pas compte que cela ruinerait le produit qu'ils espéraient vendre à travers lui. Vendre des logiciels sur le Web par l'intermédiaire de FAI, c'est comme vendre des sushis par l'intermédiaire de distributeurs automatiques.

Clients

Qui seront les clients ? Chez Viaweb, ils étaient initialement des particuliers et des petites entreprises, et je pense que ce sera la règle avec les applications basées sur le Web. Ce sont les utilisateurs qui sont prêts à essayer de nouvelles choses, en partie parce qu'ils sont plus flexibles, et en partie parce qu'ils veulent les coûts inférieurs des nouvelles technologies.

Les applications Web seront souvent la meilleure chose pour les grandes entreprises aussi (bien qu'elles soient lentes à s'en rendre compte). Le meilleur intranet est Internet. Si une entreprise utilise de véritables applications Web, le logiciel fonctionnera mieux, les serveurs seront mieux administrés et les employés auront accès au système de n'importe où.

L'argument contre cette approche repose généralement sur la sécurité : si l'accès est plus facile pour les employés, ce sera aussi pour les méchants. Certains grands commerçants étaient réticents à utiliser Viaweb parce qu'ils pensaient que les informations de carte de crédit des clients seraient plus sûres sur leurs propres serveurs. Il n'a pas été facile de faire valoir ce point diplomatiquement, mais en fait, les données étaient presque certainement plus sûres entre nos mains que les leurs. Qui peut embaucher de meilleures personnes pour gérer la sécurité, une start-up technologique dont toute l'entreprise gère des serveurs, ou un détaillant de vêtements ? Non seulement nous avions de meilleures personnes qui s'inquiétaient de la sécurité, mais nous

nous en inquiétions davantage. Si quelqu'un s'infiltrait dans les serveurs du détaillant de vêtements, cela affecterait tout au plus un commerçant, pourrait probablement être étouffé et, dans le pire des cas, pourrait faire licencier une personne. Si quelqu'un s'est fait irruption dans le nôtre, cela pourrait affecter des milliers de commerçants, finirait probablement comme une nouvelle sur CNet, et pourrait nous mettre en faillite.

Si vous voulez garder votre argent en sécurité, le gardez-vous sous votre matelas à la maison ou le mettez-vous dans une banque ? Cet argument s'applique à tous les aspects de l'administration du serveur : pas seulement la sécurité, mais aussi le temps de disponibilité, la bande passante, la gestion de la charge, les sauvegardes, etc. Notre existence dépendait de bien faire ces choses. Les problèmes de serveur étaient un grand non pour nous, comme un jouet dangereux pour un fabricant de jouets, ou une épidémie de salmonelle pour un robot culinaire.

Une grande entreprise qui utilise des applications basées sur le Web est à cette ancienne externalisation de l'informatique. Bien que cela puisse paraître, je pense que c'est généralement une bonne idée. Les entreprises sont susceptibles d'obtenir un meilleur service de cette façon que les administrateurs système internes. Les administratifs du système peuvent devenir grincheux et insensibles parce qu'ils ne sont pas directement exposés à la pression concurrentielle. Un vendeur doit traiter avec des clients, et un développeur doit traiter avec les logiciels des concurrents, mais un administrateur système, comme un vieux célibataire, a peu de forces externes pour le maintenir en ligne [11]. Chez Viaweb, nous avions beaucoup de forces externes pour nous maintenir en ligne. Les gens qui nous appelaient étaient des clients, pas seulement des collègues. Si un serveur était coincé, nous sautions. Le simple fait d'y penser me donne une poussée d'adrénaline, des années plus tard.

Ainsi, les applications Web seront généralement la bonne réponse pour les grandes entreprises aussi. Ils seront les derniers à s'en rendre compte, cependant, tout comme ils l'étaient avec les ordinateurs de bureau. Et en partie pour la même raison : cela vaudra beaucoup d'argent pour convaincre les grandes entreprises qu'elles ont besoin de quelque chose de plus cher.

Il y a toujours une tendance pour les clients riches à acheter des solutions coûteuses, même lorsque les solutions bon marché sont meilleures, parce que les personnes qui proposent des solutions coûteuses peuvent dépenser plus pour les vendre. Chez Viaweb, nous avons toujours été confrontés à cela. Nous avons perdu plusieurs commerçants haut de gamme au profit de sociétés de conseil en ligne qui les ont convaincus qu'ils seraient mieux lotis s'ils payaient un demi-million de dollars pour une boutique en ligne sur mesure sur leur propre serveur. Ils n'étaient, en règle générale, pas pariés, comme plus d'un a découvert lorsque la saison des achats de Noël est arrivée et que des charges ont augmenté sur leur serveur. Viaweb était beaucoup plus sophistiqué que ce que la plupart de ces commerçants ont obtenu, mais nous ne pouvions pas nous permettre de le leur dire. À 300 \$ par mois, nous ne pouvions pas nous permettre d'envoyer une équipe de personnes bien habillées et faisant autorité pour faire des présentations aux clients.

Parfois, nous avons joué avec l'idée d'un nouveau service appelé Viaweb Gold. Il aurait exactement les mêmes caractéristiques que notre service régulier, mais coûterait dix fois plus cher qu'il serait vendu en personne par un homme en costume. Nous n'avons jamais eu le temps d'offrir cette variante, mais je suis sûr que nous aurions pu y inscrire quelques commerçants.

Une grande partie de ce pour quoi les grandes entreprises paient en plus est le coût de leur vendre des choses chères. (Si le ministère de la Défense paie mille dollars pour des sièges de toilette, c'est en partie parce qu'il coûte cher de vendre des sièges de toilette pour mille dollars.) Et c'est l'une des raisons pour lesquelles les logiciels intranet continueront à prospérer, même si c'est probablement une mauvaise idée. C'est tout simplement plus cher. Il n'y a rien que vous puissiez faire à propos de cette énigme, donc le meilleur plan est d'aller d'abord pour les plus petits clients. Le reste viendra à temps.

Fils du serveur

L'exécution de logiciels sur le serveur n'a rien de nouveau. En fait, c'est l'ancien modèle : les applications mainframe sont toutes basées sur un serveur. Si un logiciel basé sur un serveur est une si bonne idée, pourquoi a-t-il perdu la dernière fois ? Pourquoi les ordinateurs de bureau ont-ils éclipsé les ordinateurs centraux ?

Au début, les ordinateurs de bureau ne ressemblaient pas à une grande menace. Les premiers utilisateurs étaient tous des hackers - ou des amateurs, comme on les appelait à l'époque. Ils aimaient les micro-ordinateurs parce qu'ils étaient bon marché. Pour la première fois, vous pourriez avoir votre propre ordinateur. L'expression "ordinateur personnel" fait maintenant partie de la langue, mais lorsqu'elle a été utilisée pour la première fois, elle avait un son délibérément audacieux, comme l'expression "satellite personnel" le ferait aujourd'hui.

Pourquoi les ordinateurs de bureau ont-ils pris le relais ? Principalement parce qu'ils avaient de meilleurs logiciels. Et la raison pour laquelle le logiciel de micro-ordinateur était meilleur était qu'il pouvait être écrit par de petites entreprises.

Je ne pense pas que beaucoup de gens se rendent compte à quel point les startups sont fragiles et provisoires à un stade précoce. De nombreuses startups commencent presque par accident - en tant que potes, que ce soit avec des emplois de jour ou à l'école, écrivant un prototype de quelque chose qui pourrait, s'il semble prometteur, se transformer en entreprise. À ce stade larvaire, tout obstacle important arrêtera la start-up morte sur ses rails. L'écriture d'un logiciel mainframe nécessitait trop d'engagement dès le départ. Les machines de développement étaient chères, et parce que les clients seraient de grandes entreprises, vous auriez besoin d'une force de vente impressionnante pour les leur vendre. Démarrer une start-up pour écrire un logiciel mainframe serait une entreprise beaucoup plus sérieuse que de simplement bidouiller quelque chose ensemble sur votre Apple II le soir. Et donc vous n'avez pas eu beaucoup de startups écrivant des applications mainframe.

L'arrivée des ordinateurs de bureau a inspiré beaucoup de nouveaux logiciels, car l'écriture d'applications pour eux semblait un objectif réalisable pour les startups au niveau larvaire. Le développement était bon marché, et les clients seraient des personnes individuelles que vous pouviez joindre par le biais de magasins d'informatique ou même par correspondance.

L'application qui a poussé les ordinateurs de bureau dans le grand public était VisiCalc, la première feuille de calcul. Il a été écrit par deux gars travaillant dans un grenier, et pourtant ont fait des choses qu'aucun logiciel mainframe ne pouvait faire [12]. VisiCalc était un tel progrès, en son temps, que

les gens ont acheté des Apple IIIs juste pour l'exécuter. Et c'était le début d'une tendance : les ordinateurs de bureau ont gagné parce que les startups ont écrit des logiciels pour eux.

Il semble que les logiciels sur serveur seront bons cette fois-ci, parce que les startups l'écriront. Les ordinateurs sont si bon marché maintenant que vous pouvez commencer, comme nous l'avons fait, à utiliser un ordinateur de bureau comme serveur. Les processeurs bon marché ont mangé le marché des stations de travail (vous entendez rarement le mot maintenant) et sont la plupart du chemin à travers le marché des serveurs ; les serveurs de Yahoo, qui traitent des charges aussi élevées que n'importe quelle autre sur Internet, ont tous les mêmes processeurs Intel que vous avez dans votre ordinateur de bureau. Et une fois que vous avez écrit le logiciel, tout ce dont vous avez besoin pour le vendre est un site web. Presque tous nos utilisateurs sont venus directement sur notre site par le bouche à oreille et des références dans la presse [13].

Viaweb était une start-up larvaire typique. Nous étions terrifiés à l'époque de créer une entreprise, et pendant les premiers mois, nous nous sommes réconfortés en traitant le tout comme une expérience que nous pourrions nier à tout moment. Heureusement, il y avait peu d'obstacles à l'exception des obstacles techniques. Pendant que nous écrivions le logiciel, notre serveur web était la même machine de bureau que celle que nous utilisions pour le développement, connectée au monde extérieur par une ligne commutée. Nos seules dépenses à ce stade étaient la nourriture et le loyer.

Il y a d'autant plus de raisons pour les startups d'écrire des logiciels basés sur le Web maintenant, parce que l'écriture de logiciels de bureau est devenue beaucoup moins amusante. Si vous voulez écrire des logiciels de bureau maintenant, vous le faites selon les conditions de Microsoft, en appelant leurs API et en travaillant autour de leur système d'exploitation buggy. Et si vous parvenez à écrire quelque chose qui décolle, vous constaterez peut-être que vous faisiez simplement des études de marché pour Microsoft.

Si une entreprise veut créer une plate-forme sur laquelle les startups s'appuieront, elle doit en faire quelque chose que les pirates eux-mêmes voudront utiliser. Cela signifie qu'il doit être peu coûteux et bien conçu. Le Mac était populaire auprès des pirates informatiques lorsqu'il est sorti pour la

première fois, et beaucoup d'entre eux ont écrit un logiciel pour lui [14]. Vous le voyez moins avec Windows, parce que les hackers ne l'utilisent pas. Le genre de personnes qui sont douées pour écrire des logiciels ont tendance à utiliser Linux ou FreeBSD maintenant.

Je ne pense pas que nous aurions commencé une start-up pour écrire des logiciels de bureau, parce que les logiciels de bureau doivent fonctionner sur Windows, et avant que nous ne devions écrire des logiciels pour Windows, nous devions l'utiliser. Le Web nous permet de faire une fin de course autour de Windows et de fournir des logiciels fonctionnant sous Unix directement aux utilisateurs via le navigateur. C'est une perspective libératrice, un tel que l'arrivée des PC il y a vingt-cinq ans.

Microsoft

Lorsque les ordinateurs de bureau sont arrivés, IBM était le géant que tout le monde craignait. C'est difficile à imaginer maintenant, mais je me souviens bien de ce sentiment. Maintenant, le géant effrayant est Microsoft, et je ne pense pas qu'ils soient aussi aveugles à la menace à laquelle ils sont confrontés qu'IBM. Après tout, Microsoft a délibérément construit son entreprise dans l'angle mort d'IBM.

J'ai mentionné plus tôt que ma mère n'avait pas vraiment besoin d'un ordinateur de bureau. La plupart des utilisateurs ne le font probablement pas. C'est un problème pour Microsoft, et ils le savent. Si les applications s'exécutent sur des serveurs distants, personne n'a besoin de Windows. Que va faire Microsoft ? Seront-ils en mesure d'utiliser leur contrôle du bureau pour empêcher ou contraindre cette nouvelle génération de logiciels ?

Je m'attends à ce que Microsoft développe une sorte d'hybride serveur/ordinateur de bureau, où le système d'exploitation fonctionne avec les serveurs qu'ils contrôlent. Au minimum, les fichiers seront disponibles de manière centralisée pour les utilisateurs qui le souhaitent. Je ne m'attends pas à ce que Microsoft aille jusqu'à l'extrême de faire les calculs sur le serveur, avec seulement un navigateur pour un client, s'ils peuvent l'éviter. Si vous n'avez besoin que d'un navigateur pour un client, vous n'avez pas besoin de Microsoft

sur le client, et si Microsoft ne contrôle pas le client, ils ne peuvent pas pousser les utilisateurs vers leurs applications sur serveur.

Je pense que Microsoft aura du mal à garder le génie dans la bouteille. Il y aura trop de différents types de clients pour qu'ils les contrôlent tous. Et si les applications de Microsoft ne fonctionnent qu'avec certains clients, les concurrents seront en mesure de les surpasser en proposant des applications qui fonctionnent à partir de n'importe quel client [15].

Dans un monde d'applications basées sur le Web, il n'y a pas de place automatique pour Microsoft. Ils peuvent réussir à se faire une place, mais je ne pense pas qu'ils domineront ce nouveau monde comme ils l'ont fait dans le monde des applications de bureau.

Ce n'est pas tant qu'un concurrent les fasse trébucher qu'ils se fassent trébucher eux-mêmes. Avec l'essor des logiciels basés sur le Web, ils seront confrontés non seulement à des problèmes techniques, mais aussi à leurs propres vœux pieux. Ce qu'ils doivent faire, c'est cannibaliser leur entreprise existante, et je ne peux pas les voir faire face à cela. La même détermination d'esprit qui les a amenés jusqu'ici va maintenant travailler contre eux. IBM était exactement dans la même situation, et ils ne pouvaient pas la maîtriser. IBM a fait une entrée tardive et timide dans le secteur des micro-ordinateurs parce qu'ils étaient ambivalents à l'idée de menacer leur "vache à lait", l'informatique mainframe. Microsoft sera également entravé par le fait de vouloir sauver le bureau. Une vache à lait peut être un lourd fardeau sur votre dos.

Je ne dis pas que personne ne dominera les applications basées sur le serveur. Quelqu'un finira probablement par le faire. Mais je pense qu'il y aura une bonne longue période de chaos joyeux, tout comme il y en avait dans les premiers jours des micro-ordinateurs. C'était un bon moment pour les startups. Beaucoup de petites entreprises ont prospéré, et l'ont fait en faisant des choses cool.

Les startups, mais plus encore

La start-up classique est rapide et informelle, avec peu de gens et peu d'argent. Ces quelques personnes travaillent très dur, et la technologie agrandit l'effet des décisions qu'elles prennent. S'ils gagnent, ils gagnent gros.

Dans une start-up qui écrit des applications Web, tout ce que vous associez aux startups est porté à l'extrême. Vous pouvez écrire et lancer un produit avec encore moins de personnes et encore moins d'argent. Vous devez être encore plus rapide, et vous pouvez vous en tirer en étant plus informel. Vous pouvez littéralement lancer votre produit en tant que trois gars opérant depuis d'un appartement, avec un serveur placé chez un FAI. Nous l'avons fait.

Au fil du temps, les équipes sont devenues plus petites, plus rapides et plus informelles. En 1960, le développement de logiciels signifiait une salle pleine d'hommes avec des lunettes à monture en corne et d'étroites cravates noires, écrivant de manière indurelignée dix lignes de code par jour sur des formulaires de codage IBM. En 1980, il s'agissait d'une équipe de huit à dix personnes portant un jean au bureau et tapant dans les VT100. Maintenant, il y a quelques gars assis dans un salon avec des ordinateurs portables. (Les jeans ne sont pas le dernier mot en matière d'informalité.)

Les startups sont stressantes, et cela, malheureusement, est également porté à l'extrême avec les applications Web. De nombreuses sociétés de logiciels, en particulier au début, ont des périodes où les développeurs dormaient sous leur bureau et ainsi de suite. Ce qui est alarmant à propos des logiciels basés sur le Web, c'est qu'il n'y a rien pour empêcher que cela devienne la valeur par défaut. Les histoires sur le fait de dormir sous les bureaux se terminent généralement : enfin, nous l'avons expédié, et nous sommes tous rentrés à la maison et avons dormi pendant une semaine. Les logiciels basés sur le Web ne sont jamais expédiés. Vous pouvez travailler 16 heures par jour aussi longtemps que vous le souhaitez. Et parce que vous le pouvez, et que vos concurrents le peuvent, vous avez tendance à être forcés de le faire. Vous pouvez, donc vous devez. C'est la loi de Parkinson qui fonctionne à l'envers.

Le pire n'est pas les heures, mais la responsabilité. Les professionnels et les administrateurs système ont traditionnellement chacun leurs propres

préoccupations. Les programmeurs s'inquiètent des bugs, et les administrateurs système s'inquiètent de l'infrastructure. Les programmeurs peuvent passer une longue journée à se plonger jusqu'aux coudes dans le code source, mais à un moment donné, ils peuvent rentrer chez eux et l'oublier. Les administrateurs du système ne laissent jamais tout à fait le travail derrière eux, mais lorsqu'ils sont téléphonés à 4 heures du matin, ils n'ont généralement pas à faire quoi que ce soit de très compliqué. Avec les applications Web, ces deux types de stress se combinent. Les programmeurs deviennent administrateurs système, mais sans les limites bien définies qui rendent habituellement le travail supportable.

Chez Viaweb, nous avons passé les six premiers mois à écrire des logiciels. Nous avons travaillé les longues heures habituelles d'une start-up précoce. Dans une entreprise de logiciels de bureau, cela aurait été la partie la plus difficile, mais on l'a ressenti comme des vacances par rapport à la phase suivante, lorsque nous avons emmené les utilisateurs sur notre serveur. Le deuxième plus grand avantage de la vente de Viaweb à Yahoo (après l'argent) a été de pouvoir rejeter la responsabilité ultime de l'ensemble sur les épaules d'une grande entreprise.

Les logiciels de bureau obligent les utilisateurs à devenir des administrateurs système. Les logiciels basés sur le Web obligent les programmeurs à le faire. Il y a moins de stress au total, mais plus pour les programmeurs. Ce n'est pas nécessairement une mauvaise nouvelle. Si vous êtes une start-up en concurrence avec une grande entreprise, c'est une bonne nouvelle [16]. Les applications Web offrent un moyen simple de surpasser vos concurrents. Aucune start-up n'en demande plus.

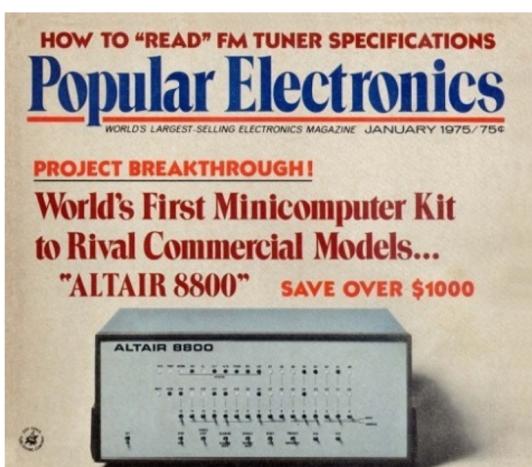
Juste assez bon

Une chose qui pourrait vous dissuader d'écrire des applications basées sur le Web est la nullité des pages Web en tant qu'interface utilisateur. C'est un problème, je l'avoue. Il y avait quelques choses que nous aurions *vraiment* aimé ajouter à HTML et HTTP. Ce qui compte, cependant, c'est que les pages Web soient juste assez bonnes.

Il y a ici un parallèle avec les premiers micro-ordinateurs. Les processeurs de ces machines n'étaient pas destinés à être les processeurs des

ordinateurs. Ils ont été conçus pour être utilisés dans des choses comme les feux de circulation. Mais des gars comme Ed Roberts, qui a conçu l'Altair, se sont rendu compte qu'ils étaient juste assez bons. Vous pourriez combiner l'une de ces puces avec de la mémoire (256 octets dans le premier Altair) et des commutateurs sur le panneau avant, et vous auriez un ordinateur fonctionnel. Être capable d'avoir son propre ordinateur était si excitant qu'il y avait beaucoup de gens qui voulaient l'acheter, aussi limités soient-ils.

Les pages Web n'ont pas été conçues pour être une interface utilisateur pour les applications, mais elles sont juste assez bonnes. Et pour un nombre important d'utilisateurs, les logiciels que vous pouvez utiliser à partir de n'importe quel navigateur seront une victoire en soi pour l'emporter sur toute maladresse dans l'interface utilisateur. Peut-être que vous ne pouvez pas écrire la plus belle feuille de calcul en utilisant HTML, mais vous pouvez écrire une feuille de calcul que plusieurs personnes peuvent utiliser simultanément à partir de différents endroits sans logiciel client spécial, ou qui peut ajouter des flux de données en direct, ou qui peut vous page lorsque certaines conditions sont déclenchées. Plus important encore, vous pouvez écrire de nouveaux types d'applications qui n'ont même pas encore de noms. VisiCalc n'était pas simplement une version micro-ordinateur d'une application mainframe, après tout, c'était un nouveau type d'application.



Électronique populaire, janvier 1975 (détail).

Bien sûr, les applications basées sur le serveur n'ont pas besoin d'être basées sur le Web. Vous pourriez avoir un autre type de client. Mais je suis à peu près sûr que c'est une mauvaise idée. Ce serait très pratique si vous pouviez supposer que tout le monde installe votre client - si pratique que vous pourriez

facilement vous convaincre qu'ils le feraient tous. Mais s'ils ne le font pas, vous êtes démunis.

Parce que le logiciel basé sur le Web ne suppose rien du client, il fonctionnera partout où le Web fonctionne. C'est un grand avantage déjà prêt, et l'avantage augmentera à mesure que de nouveaux appareils web prolifèrent. Les utilisateurs vous aimeront parce que votre logiciel fonctionne tout simplement, et votre vie sera plus facile parce que vous n'aurez pas à le modifier pour chaque nouveau client [17].

J'ai l'impression d'avoir observé l'évolution du Web aussi étroitement que n'importe qui, et je ne peux pas prédire ce qui va se passer avec les clients. La convergence arrive probablement, mais où ?

Comment tout cela se passera-t-il ? Je ne sais pas. Et vous n'avez pas besoin de savoir si vous parlez sur des applications Web. Personne ne peut briser cela sans casser la navigation. Le Web n'est peut-être pas le seul moyen de fournir des logiciels, mais c'est un logiciel qui fonctionne maintenant et qui continuera à fonctionner pendant longtemps. Les applications Web sont bon marché à développer, et faciles à fournir même pour la plus petite start-up. C'est beaucoup de travail, et d'un genre particulièrement stressant, mais cela ne fait qu'améliorer les chances pour les startups.

Pourquoi pas

E. B. White s'est amusé à apprendre d'un ami agriculteur que de nombreuses clôtures électrifiées n'ont pas de courant qui les traverse. Les vaches apprennent apparemment à rester loin d'elles, et après cela, vous n'avez pas besoin du courant. « Levez-vous, les vaches ! » Il a écrit. « Prenez votre liberté pendant que les despotes ronflent ! »

Si vous êtes un hacker qui a pensé un jour démarrer une start-up, il y a probablement deux choses qui vous empêchent de le faire. La première est que vous ne savez rien sur les affaires. L'autre est que vous avez peur de la concurrence. Aucune de ces clôtures n'a de courant.

Il n'y a que deux choses que vous devez savoir sur les affaires : construire quelque chose que les utilisateurs aiment et faire plus que ce que vous dépensez. Si vous avez bien compris ces deux, vous aurez une longueur d'avance sur la plupart des startups. Vous pouvez comprendre le reste au fur et à mesure.

Vous ne gagnez peut-être pas au début plus que ce que vous dépensez, mais tant que l'écart se ferme assez rapidement, tout ira bien. Si vous commencez sous-financé, cela encouragera au moins une habitude de frugalité. Moins vous dépensez, plus il est facile de faire plus que vous ne dépensez. Heureusement, il peut être très bon marché de lancer une application Web. Nous avons lancé moins de 10 000 \$, et ce serait encore moins cher aujourd'hui. Nous avons dû dépenser des milliers sur un serveur, et des milliers d'autres pour obtenir SSL. (La seule entreprise qui vendait des logiciels SSL à ce moment là était Netscape.) Maintenant, vous pouvez louer un serveur beaucoup plus puissant, avec SSL inclus, pour moins que ce que nous avons payé pour la seule bande passante. Vous pourriez lancer une application Web maintenant pour moins que le coût d'une chaise de bureau de luxe.

En ce qui concerne la construction de quelque chose que les utilisateurs aiment, voici quelques conseils généraux. Commencez par faire quelque chose de propre et de simple que vous voudriez utiliser vous-même. Obtenez une version 1.0 rapidement, puis continuez à améliorer le logiciel, en écoutant attentivement les utilisateurs comme vous le faites. Le client a toujours raison, mais différents clients ont raison sur des choses différentes ; les utilisateurs les moins sophistiqués vous montrent ce dont vous avez besoin pour simplifier et clarifier, et les plus sophistiqués vous disent quelles fonctionnalités vous devez ajouter. La meilleure chose que le logiciel puisse être est facile, mais la façon de le faire est d'obtenir les bonnes valeurs par défaut, et non de limiter les choix des utilisateurs. Ne soyez pas complaisant si le logiciel de vos concurrents est boiteux ; la norme à laquelle comparer votre logiciel est ce qu'il pourrait être, pas ce que vos concurrents actuels ont. Utilisez votre logiciel vous-même, tout le temps. Viaweb était censé être un créateur de boutique en ligne, mais nous l'avons également utilisé pour créer notre propre site. N'écoutez pas les gens du marketing, les concepteurs ou les chefs de produits simplement à cause de leurs titres de poste. S'ils ont de bonnes idées, utilisez-les, mais c'est à vous de décider ; le logiciel doit être conçu par des hackers qui comprennent le design, pas par des concepteurs qui en savent un peu plus sur le logiciel. Si vous ne

pouvez pas concevoir un logiciel aussi bien que l'implémenter, ne démarrez pas une start-up.

Parlons maintenant de la concurrence. Ce dont vous avez peur, ce n'est probablement pas des groupes de hackers comme vous, mais des entreprises réelles, avec des bureaux et des plans d'affaires et des vendeurs et ainsi de suite, n'est-ce pas ? Eh bien, ils ont plus peur de vous que vous ne l'êtes d'eux, et ils ont raison. Il est beaucoup plus facile pour quelques hackers de découvrir comment louer des bureaux ou embaucher des vendeurs que pour une entreprise de toute taille de faire écrire un logiciel. J'ai été des deux côtés, et je sais. Lorsque Viaweb a été acheté par Yahoo, je me suis soudainement retrouvé à travailler pour une grande entreprise, et c'était comme essayer de courir à travers l'eau jusqu'à la taille.



Bill Gates, 1977.

Je ne veux pas dénigrer Yahoo. Ils avaient de bons hackers, et la haute direction était de vrais butt-kickers. Pour une grande compagnie, ils étaient exceptionnels. Mais ils n'étaient encore qu'environ un dixième aussi productifs qu'une petite start-up. Aucune grande entreprise ne peut faire beaucoup mieux que cela. Ce qui est effrayant à propos de Microsoft, c'est qu'une entreprise aussi grande peut développer des logiciels. Ils sont comme une montagne qui peut marcher.

Ne vous faites pas intimider. Vous pouvez faire autant que Microsoft ne peut pas faire et ce que vous ne pouvez pas faire et que Microsoft peut. Et personne ne peut vous arrêter. Vous n'avez pas besoin de demander la permission à qui que ce soit pour développer des applications basées sur le Web.

Vous n'avez pas besoin de faire des offres de licence, ou d'obtenir de l'espace sur les étagères dans les magasins de détail, ou de s'époumoner pour que votre application soit fournie avec le système d'exploitation. Vous pouvez fournir des logiciels directement au navigateur, et personne ne peut s'interposer entre vous et les utilisateurs potentiels sans les empêcher de naviguer sur le Web.

Vous ne le croyez peut-être pas, mais je vous le promets, Microsoft a peur de vous. Les cadres intermédiaires complaisants ne le sont peut-être pas, mais Bill Gates l'est, parce qu'il était vous une fois, en 1975, la dernière fois qu'une nouvelle façon de livrer des logiciels est apparue.

Chapitre 6

Comment Créer de la Valeur

Si vous vouliez devenir riche, comment le feriez-vous ? Je pense que votre meilleur pari serait de démarrer ou de rejoindre une start-up. C'est un moyen fiable de s'enrichir depuis des centaines d'années. Le mot « start-up » date des années 1960, mais ce qui se passe dans l'un d'eux est très similaire aux voyages commerciaux du Moyen Âge.

Les start-ups impliquent généralement la technologie, à tel point que l'expression « start-up de haute technologie » est presque redondante. Une start-up est une petite entreprise qui s'attaque à un problème technique difficile.

Beaucoup de gens s'enrichissent en ne sachant rien de plus que cela. Vous n'avez pas besoin de connaître la physique pour être un bon lanceur. Mais je pense que cela pourrait vous donner un avantage pour comprendre les principes sous-jacents. Pourquoi les start-ups doivent-elles être petites ? Une start-up cessera-t-elle inévitablement d'être une start-up à mesure qu'elle grandit ? Et pourquoi travaillent-ils si souvent au développement de nouvelles technologies ? Pourquoi y a-t-il tant de start-ups qui vendent de nouveaux médicaments ou des logiciels informatiques, et aucune ne vend de l'huile de maïs ou du détergent à lessive ?

La proposition

Sur le plan économique, vous pouvez penser à une start-up comme un moyen de comprimer toute votre vie professionnelle en quelques années. Au lieu de travailler à faible intensité pendant quarante ans, vous travaillez aussi dur que possible pendant quatre ans. Cela paie particulièrement bien dans la technologie, où vous gagnez une prime pour travailler rapidement.

Voici un bref aperçu de la proposition économique. Si vous êtes un bon hacker au milieu de la vingtaine, vous pouvez obtenir un emploi en payant environ 80 000 \$ par an. Donc, en moyenne, un tel hacker doit être capable de faire au moins 80 000 \$ de travail par an pour l'entreprise juste pour atteindre le seuil de rentabilité. Vous pourriez probablement travailler deux fois plus

d'heures qu'un employé de l'entreprise, et si vous vous concentrez, vous pouvez probablement obtenir trois fois plus de fait en une heure [1]. Vous devriez avoir plusieurs de deux, au moins, en éliminant la traînée du cadre intermédiaire aux cheveux pointues qui serait votre patron dans une grande entreprise. Ensuite, il y a un autre multiple : à quel point êtes-vous plus intelligent que ce que votre description de poste s'attend à ce que vous soyez ? Supposons un autre multiple de trois. Combinez tous ces multiplicateurs, et je prétends que vous pourriez être 36 fois plus productif que ce que vous êtes censé être dans un emploi d'entreprise aléatoire [2]. Si un assez bon hacker vaut 80 000 \$ par an une grande entreprise, alors un hacker intelligent qui travaille très dur sans aucune connerie d'entreprise pour le ralentir devrait être en mesure de faire un travail qui vaudrait à peu près 3 millions de \$ par an

Comme tous les calculs à l'emporte-pièce, celui-ci a une grande marge de manœuvre. Je n'essayerais pas de défendre les chiffres réels. Mais je tiens à la structure du calcul. Je ne prétends pas que le multiplicateur est précisément de 36, mais il est certainement supérieur à 10, et probablement rarement aussi élevé que 100.

Si 3 millions de dollars par an semble élevé, rappelez-vous que nous parlons du cas limite : le cas où non seulement vous n'avez pas de temps libre, mais que vous travaillez si dur que vous mettez votre santé en danger.

Les startups ne sont pas magiques. Ils ne changent pas les lois de la création de richesse. Ils ne représentent qu'un point à l'extrémité de la courbe. Il y a une loi sur la conservation à l'œuvre ici : si vous voulez gagner un million de dollars, vous devez supporter un million de dollars de douleur. Par exemple, une façon de gagner un million de dollars serait de travailler pour le bureau de poste toute votre vie et d'économiser chaque centime de votre salaire. Imaginez le stress de travailler pour le bureau de poste pendant cinquante ans. Dans une start-up, vous compresserez tout ce stress en trois ou quatre ans. Vous avez tendance à obtenir un certain rabais en vrac si vous achetez le mal économique mais vous ne pouvez pas échapper à la loi fondamentale sur la conservation. Si le démarrage d'une start-up était facile, tout le monde le ferait.

Des millions, pas des milliards

Si 3 millions de dollars par an semblent élevés à certaines personnes, ils sembleront faibles à d'autres. Trois *millions* ? Comment puis-je devenir milliardaire, comme Bill Gates ?

Alors, laissons Bill Gates de côté tout de suite. Ce n'est pas une bonne idée d'utiliser des riches célèbres comme exemples, car la presse n'écrit que sur les plus riches, et ceux-ci ont tendance à être des valeurs aberrantes. Bill Gates est un homme intelligent, déterminé et travailleur, mais vous avez besoin de plus que cela pour gagner autant d'argent que lui. Vous devez également avoir beaucoup de chance.

Il y a un grand facteur aléatoire dans le succès de toute entreprise. Donc, les gars que vous finissez par lire dans les journaux sont ceux qui sont très intelligents, totalement dévoués et qui gagnent à la loterie. Il est certain que Bill est intelligent et dévoué, mais il se trouve que Microsoft a également été le bénéficiaire de l'une des erreurs les plus spectaculaires de l'histoire des affaires : l'accord de licence pour DOS. Il ne fait aucun doute que Bill a fait tout ce qu'il pouvait pour orienter IBM dans cette erreur, et il a fait un excellent travail pour l'exploiter, mais s'il y avait eu une personne avec un cerveau du côté d'IBM, l'avenir de Microsoft aurait été très différent. À ce stade, Microsoft avait peu d'influence sur IBM. Ils étaient effectivement un fournisseur de composants. Si IBM avait eu besoin d'une licence exclusive, comme ils auraient dû, Microsoft aurait quand même signé l'accord. Cela aurait quand même signifié beaucoup d'argent pour eux, et IBM aurait facilement pu obtenir un système d'exploitation ailleurs.

Au lieu de cela, IBM a fini par utiliser toute sa puissance sur le marché pour donner à Microsoft le contrôle de la norme PC. À partir de ce moment, tout ce que Microsoft avait à faire était d'exécuter. Ils n'ont jamais eu à parier l'affaire sur une décision audacieuse. Tout ce qu'ils avaient à faire était de jouer dur avec les titulaires de licence et de copier des produits plus innovants raisonnablement et rapidement.

Si IBM n'avait pas commis cette erreur, Microsoft aurait toujours été une entreprise prospère, mais elle n'aurait pas pu grandir aussi rapidement. Bill

Gates serait riche, mais il serait quelque part près du bas du Forbes 400 avec les autres gars de son âge.

Il y a beaucoup de façons de s'enrichir, et cet essai n'est qu'une seule d'entre elles. Cet essai explique comment gagner de l'argent en créant de la richesse et en étant payé pour cela. Il existe de nombreuses autres façons d'obtenir de l'argent, y compris le hasard, la spéculation, le mariage, l'héritage, le vol, l'extorsion, la fraude, le monopole, la greffe, le lobbying, la contrefaçon et la prospection. La plupart des plus grandes fortunes ont probablement impliqué plusieurs d'entre elles.

L'avantage de créer de la richesse, en tant que moyen de devenir riche, n'est pas seulement que c'est plus légitime (de nombreuses autres méthodes sont maintenant illégales), mais que c'est plus simple. Il suffit de faire quelque chose que les gens veulent.

L'argent n'est pas la richesse

Si vous voulez créer de la richesse, cela vous aidera à comprendre ce que c'est. La richesse n'est pas la même chose que l'argent [3]. La richesse est aussi ancienne que l'histoire humaine. Beaucoup plus vieille, en fait ; les fourmis ont de la richesse. L'argent est une invention relativement récente.

La richesse est la chose fondamentale. La richesse est ce que nous voulons : de la nourriture, des vêtements, des maisons, des voitures, des gadgets, des voyages dans des endroits intéressants, et ainsi de suite. Vous pouvez avoir de la richesse sans avoir d'argent. Si vous aviez une machine magique qui, sur commande, vous fait une voiture ou vous prépare le dîner ou fait votre lessive, ou faire quoi que ce soit d'autre que vous vouliez, vous n'auriez pas besoin d'argent. Alors que si vous étiez au milieu de l'Antarctique, où il n'y a rien à acheter, peu importe combien d'argent vous aviez.

La richesse est ce que vous voulez, pas l'argent. Mais si la richesse est la chose la plus importante, pourquoi tout le monde parle-t-il de gagner de l'argent ? C'est une sorte de sténographie : l'argent est un moyen de déplacer la richesse, et en pratique, il est généralement interchangeable. Mais ce n'est pas la même chose, et à moins que vous ne prévoyez de devenir riche en faisant de la

contrefaçon, parler de gagner de l'argent peut rendre plus difficile de comprendre comment gagner de l'argent.

L'argent est un effet secondaire de la spécialisation. Dans une société spécialisée, la plupart des choses dont vous avez besoin, vous ne pouvez pas les faire pour vous-même. Si vous voulez une pomme de terre ou un crayon ou un endroit où vivre, vous devez l'obtenir de quelqu'un d'autre.

Comment faites-vous pour que la personne qui cultive les pommes de terre vous en donne ? En lui donnant quelque chose qu'il veut en retour. Mais vous ne pouvez pas aller très loin en échangeant les choses directement avec les personnes qui en ont besoin. Si vous faites des violons, et qu'aucun des agriculteurs locaux n'en veut un, comment allez-vous manger ?

Les sociétés de solutions trouvent qu'au fur et à mesure qu'elles se spécialisent, il est de transformer le commerce en un processus en deux étapes. Au lieu d'échanger des violons directement contre des pommes de terre, vous échangez des violons contre, par exemple, de l'argent, que vous pouvez ensuite échanger à nouveau contre tout ce dont vous avez besoin. Les choses intermédiaires - le moyen d'échange - peuvent être tout ce qui est rare et portable. Historiquement, les métaux ont été les plus courants, mais récemment, nous avons utilisé un moyen d'échange, appelé le dollar, qui n'existe pas physiquement. Il fonctionne comme un moyen d'échange, cependant, parce que sa rareté est garantie par le Gouvernement américain.

L'avantage d'un moyen d'échange est qu'il fait fonctionner le commerce. L'inconvénient est qu'il a tendance à obscurcir ce que le commerce signifie réellement. Les gens pensent que ce qu'une entreprise fait, c'est gagner de l'argent. Mais l'argent n'est que l'étape intermédiaire - juste un raccourci - pour tout ce que les gens veulent. Ce que la plupart des entreprises font vraiment, c'est gagner de la richesse. Ils font quelque chose que les gens veulent [4].

L'erreur de la tarte

Un nombre surprenant de personnes conservent dès l'enfance l'idée qu'il y a une quantité fixe de richesse dans le monde. Il y a, dans toute famille normale, une somme *d'argent* fixe à tout moment. Mais ce n'est pas la même chose.

Lorsque l'on parle de richesse dans ce contexte, elle est souvent décrite comme une tarte. « Vous ne pouvez pas agrandir la tarte », disent les politiciens. Lorsque vous parlez du montant d'argent sur le compte bancaire d'une famille, ou du montant disponible pour un gouvernement à partir d'un an de recettes fiscales, c'est vrai. Si une personne obtient plus, quelqu'un d'autre doit obtenir moins.

Je me souviens avoir cru, quand j'étais enfant, que si quelques personnes riches avaient tout l'argent, cela laisserait moins pour tout le monde. Beaucoup de gens semblent continuer à croire quelque chose comme ça jusqu'à l'âge adulte. Cette erreur est généralement là en arrière-plan lorsque vous entendez quelqu'un parler de la façon dont X pour cent de la population a Y pour cent de la richesse. Si vous prévoyez de démarrer une start-up, que vous vous en rendiez compte ou non, vous prévoyez de réfuter l'erreur de la tarte.

Ce qui égare les gens ici, c'est l'abstraction de l'argent. L'argent n'est pas la richesse. C'est juste quelque chose que nous utilisons pour déplacer la richesse. Donc, bien qu'il puisse y avoir, à certains moments spécifiques (comme votre famille, ce mois-ci), un montant fixe d'argent disponible pour échanger avec d'autres personnes contre des choses que vous voulez, il n'y a pas de montant fixe de richesse dans le monde. Vous pouvez gagner plus de richesse. La richesse a été créée et détruite (mais en équilibre, créée) pour toute l'histoire de l'humanité.

Supposons que vous possédez une vieille voiture déglinguée. Au lieu de vous asseoir sur vos fesses l'été prochain, vous pourriez passer du temps à remettre votre voiture en parfait état. Ce faisant, vous créez de la richesse. Le monde est - et vous l'êtes spécifiquement - une vieille voiture immaculée, plus riche. Et pas seulement d'une manière métaphorique. Si vous vendez votre voiture, vous en obtiendrez plus.

En restaurant votre vieille voiture, vous vous êtes rendu plus riche. Vous n'avez rendu personne d'autre plus pauvre. Il n'y a donc évidemment pas de tarte fixe. Et en fait, quand vous le regardez de cette façon, vous vous demandez pourquoi quelqu'un penserait qu'il y en avait [5].

Les enfants savent, sans le savoir, qu'ils peuvent créer de la richesse. Si vous avez besoin de donner un cadeau à quelqu'un et que vous n'avez pas d'argent, vous en faites un. Mais les enfants sont si mauvais pour faire des choses qu'ils considèrent les cadeaux faits maison comme une sorte de chose distincte, inférieure à acheter en magasin - une simple expression de la pensée proverbiale qui compte. Et en effet, les cendriers grumeleux que nous avons fabriqués pour nos parents n'avaient pas beaucoup de marché de revente.

Artisans

Les personnes les plus susceptibles de comprendre que la richesse peut être créée sont celles qui sont bonnes pour faire des choses : les artisans. Leurs objets faits à la main deviennent ceux achetés en magasin. Mais avec la montée de l'industrialisation, il y a de moins en moins d'artisans. L'un des plus grands groupes restants est celui des programmeurs informatiques.

Un programmeur peut s'asseoir devant un ordinateur et créer de la richesse. Un bon logiciel est, en soi, une chose précieuse. Il n'y a pas de fabrication pour confondre la question. Les caractères que vous tapez sont un produit complet et fini. Si quelqu'un s'asseyait et écrivait un navigateur Web qui n'était pas nul (une bonne idée, soit dit en passant), le monde serait d'autant plus riche.

Tout le monde dans une entreprise travaille ensemble pour créer de la richesse, dans le sens de faire plus de choses que les gens veulent. Beaucoup d'employés (par exemple, les personnes de la salle du courrier ou du service du personnel) travaillent à un seul retrait de la fabrication réelle des choses. Pas les programmeurs. Ils pensent littéralement le produit, une ligne à la fois. Et il est donc plus clair pour les programmeurs que la richesse est quelque chose qui est faite, plutôt que d'être distribuée, comme des tranches de tarte, par un papa imaginaire.

Il est également évident pour les programmeurs qu'il existe d'énormes variations dans le rythme de création de la richesse. Chez Viaweb, nous avions un programmeur qui était une sorte de monstre de productivité. Je me souviens d'avoir regardé ce qu'il a fait un long jour et d'estimer qu'il avait ajouté plusieurs centaines de milliers de dollars à la valeur marchande de l'entreprise. Un grand

programmeur, sur un rouleau, pourrait créer un million de dollars de richesse en quelques semaines. Un programmeur médiocre au cours de la même période générera une richesse nulle ou même nulle (par exemple en introduisant des bugs).

C'est pourquoi tant de meilleurs programmeurs sont des libertaires. Dans notre monde, vous coulez ou vous nagez, et il n'y a pas d'excuses. Lorsque ceux qui sont loin de la création de richesse - étudiants de premier cycle, journalistes, politiciens - entendent que les 5 % les plus riches de la population ont la moitié de la richesse totale, ils ont tendance à penser à l'injustice ! Un programmeur expérimenté serait plus susceptible de penser que c'est tout ? Les 5 % les meilleurs programmeurs écrivent probablement 99 % des bons logiciels.

La richesse peut être créée sans être vendue. Les scientifiques, jusqu'à ce que ce soit au moins, ont effectivement fait don de la richesse qu'ils ont créée. Nous sommes tous plus riches pour connaître la pénicilline, parce que nous sommes moins susceptibles de mourir d'infections. La richesse est tout ce que les gens veulent, et ne pas mourir est certainement quelque chose que nous voulons. Les hackers donnent souvent leur travail en écrivant des logiciels open source que tout le monde peut utiliser gratuitement. Je suis beaucoup plus riche pour le système d'exploitation FreeBSD, que j'exécute sur l'ordinateur que j'utilise actuellement, tout comme Yahoo, qui l'exécute sur tous leurs serveurs.

Ce qu'est un travail

Dans les pays industrialisés, les gens appartiennent à une institution ou à une autre au moins jusqu'à la vingtaine. Après toutes ces années, vous vous habituez à l'idée d'appartenir à un groupe de personnes qui se lèvent toutes le matin, vont dans un ensemble de bâtiments et font des choses qu'elles n'aiment pas, normalement, faire. L'appartenance à un tel groupe fait partie de votre identité : nom, âge, rôle, institution. Si vous devez vous présenter, ou si quelqu'un d'autre vous décrit, ce sera quelque chose comme, John Smith, 10 ans, un étudiant de telle ou telle école primaire, ou John Smith, 20 ans, un étudiant de telle ou telle université.

Lorsque John Smith termine ses études, on s'attend à ce qu'il trouve un emploi. Et ce que trouver un emploi semble signifier, c'est rejoindre une autre

institution. A première vue, cela ressemble beaucoup à l'université. Vous choisissez les entreprises pour lesquelles vous voulez travailler et postulez pour les rejoindre. Si quelqu'un vous apprécie, vous devenez membre de ce nouveau groupe. Vous vous levez le matin et allez dans un nouvel ensemble de bâtiments, et vous faites des choses que vous n'aimez pas normalement faire. Il y a quelques différences : la vie n'est pas aussi amusante, et vous êtes payé, au lieu de payer, comme vous l'avez fait à l'université. Mais les similitudes sont plus grandes que les différences. John Smith est maintenant John Smith, 22 ans, développeur de logiciels dans telle ou telle société.

En fait, la vie de John Smith a changé plus qu'il ne le pense. Sur le plan social, une entreprise ressemble beaucoup à l'université, mais plus vous vous enfoncez dans la réalité sous-jacente, plus elle devient différente.

Ce qu'une entreprise fait, et doit faire si elle veut continuer à exister, c'est gagner de l'argent. Et la façon dont la plupart des entreprises gagnent de l'argent est de créer de la richesse. Les entreprises peuvent être si spécialisées que cette clarté est dissimulée, mais ce ne sont pas seulement les entreprises manufacturières qui créent de la richesse. Une grande composante de la richesse est l'emplacement. Vous vous souvenez de cette machine magique qui pourrait vous faire des voitures et vous cuisiner le dîner et ainsi de suite ? Il ne serait pas si utile qu'il livre votre dîner à un endroit aléatoire en Asie centrale. Si la richesse signifie ce que les gens veulent, les entreprises qui déplacent les choses créent également de la richesse. Idem pour de nombreux autres types d'entreprises qui ne font rien de physique. Presque toutes les entreprises existent pour faire quelque chose que les gens veulent.

Et c'est aussi ce que vous faites lorsque vous allez travailler pour une entreprise. Mais ici, il y a une autre couche qui a tendance à obscurcir la réalité sous-jacente. Dans une entreprise, le travail que vous faites est calculé en moyenne avec celui de beaucoup d'autres personnes. Vous ne savez peut-être même pas que vous faites quelque chose que les gens veulent. Votre contribution peut être indirecte. Mais l'entreprise dans son ensemble doit donner aux gens quelque chose qu'ils veulent, ou ils ne gagneront pas d'argent. Et s'ils vous paient x dollars par an, alors en moyenne, vous devez contribuer au moins x dollars par an de travail, sinon l'entreprise dépensera plus qu'elle ne gagne et mettra la clé sous la porte.

Quelqu'un diplômé de l'université pense, et on lui dit, qu'il a besoin de trouver un emploi, comme si l'important était de devenir membre d'une institution. Une façon plus directe de le dire serait : vous devez commencer à faire quelque chose que les gens veulent. Vous n'avez pas besoin de rejoindre une entreprise pour le faire. Tout ce qu'une entreprise est, c'est un groupe de personnes qui travaillent ensemble pour faire quelque chose que les gens veulent. C'est faire quelque chose que les gens veulent qui compte, ne pas rejoindre le groupe [6].

Pour la plupart des gens, le meilleur plan est probablement d'aller travailler pour une entreprise existante. Mais c'est une bonne idée de comprendre ce qui se passe lorsque vous faites cela. Un travail signifie faire quelque chose que les gens veulent, en moyenne avec tout le monde dans cette entreprise.

Travailler plus dur

Cette moyenne devient un problème. Je pense que le plus gros problème qui touche les grandes entreprises est la difficulté d'attribuer une valeur au travail de chaque personne. Pour la plupart, ils pratiquent le "punt". Dans une grande entreprise, vous recevez un salaire assez prévisible pour avoir travaillé assez dur. On s'attend à ce que vous ne soyiez pas manifestement incompétent ou paresseux, mais on ne s'attend pas à ce que vous consaciez toute votre vie à votre travail.

Il s'avère, cependant, qu'il y a des économies d'échelle dans la partie de votre vie que vous consacrez à votre travail. Dans le bon type d'entreprise, quelqu'un qui se consacre vraiment au travail pourrait générer dix ou même cent fois plus de richesse qu'un employé moyen. Un programmeur, par exemple, au lieu de maintenir et de mettre à jour un logiciel existant, pourrait écrire un tout nouveau logiciel et, avec lui, créer une nouvelle source de revenus.

Les entreprises ne sont pas mises en place pour récompenser les personnes qui veulent le faire. Vous ne pouvez pas aller voir votre patron et lui dire que "j'aimerais commencer à travailler dix fois plus dur, alors voulez-vous me payer dix fois plus ?" D'une part, la fiction officielle est que vous travaillez

déjà aussi dur que vous le pouvez. Mais un problème plus grave est que l'entreprise n'a aucun moyen de mesurer la valeur de votre travail.

Les vendeurs sont une exception. Il est facile de mesurer le montant des revenus qu'ils génèrent, et ils en sont généralement payés en pourcentage. Si un vendeur veut travailler plus dur, il peut simplement commencer à le faire, et il sera automatiquement payé proportionnellement plus.

Outre la vente, il existe un autre poste où les grandes entreprises peuvent embaucher des personnes de premier ordre : les postes de direction. Et ce, pour la même raison : leurs performances peuvent être mesurées. Les cadres supérieurs sont responsables des performances de l'ensemble de l'entreprise. Étant donné que les performances d'un employé ordinaire ne peuvent généralement pas être mesurées, on n'attend pas de lui qu'il fasse plus qu'un solide effort. Alors que la haute direction, comme les vendeurs, doit rendre des comptes avec des chiffres. Le PDG d'une entreprise qui fait faillite ne peut pas plaider qu'il a fait de gros efforts. Si la compagnie va mal, c'est qu'il a mal fait les choses.

Une entreprise qui pourrait payer tous ses employés de manière aussi simple connaîtrait un énorme succès. De nombreux employés travailleraient plus dur s'ils pouvaient être payés pour cela. Plus important encore, une telle entreprise attirerait des gens qui voudraient travailler particulièrement dur. Il écraserait ses concurrents.

Malheureusement, les entreprises ne peuvent pas payer tout le monde comme des vendeurs. Les vendeurs travaillent seuls. Le travail de la plupart des employés est enchevêtré. Supposons qu'une entreprise fabrique une sorte de gadget grand public. Les ingénieurs construisent un gadget fiable avec toutes sortes de nouvelles fonctionnalités ; les concepteurs industriels conçoivent un bel étui pour cela ; puis les gens du marketing convainquent tout le monde que c'est quelque chose qu'ils doivent avoir. Comment savoir combien des ventes de gadgets sont dues aux efforts de chaque groupe ? Ou, d'ailleurs, combien est dû aux créateurs de gadgets passés qui ont donné à l'entreprise une réputation de qualité ? Il n'y a aucun moyen de démêler toutes leurs contributions. Même si vous pouviez lire l'esprit des consommateurs, vous constateriez que ces facteurs étaient tous ensemble flous.

Si vous voulez aller plus vite, c'est un problème d'avoir votre travail enchevêtré avec un grand nombre d'autres personnes. Dans un grand groupe, votre performance n'est pas mesurable séparément - et le reste du groupe vous ralentit.

Mesure et effet de levier

Pour devenir riche, vous devez vous mettre dans une situation avec deux choses, la mesure et l'effet de levier. Vous devez être dans une position où votre performance peut être mesurée, ou il n'y a aucun moyen d'être payé plus en faisant plus. Et vous devez avoir un effet de levier, en ce sens que les décisions que vous prenez ont un grand effet.

La mesure seule ne suffit pas. Un exemple de travail avec mesure mais pas d'effet de levier est de faire du travail à la pièce dans un atelier de misère. Votre performance est mesurée et vous êtes payé en conséquence, mais vous n'avez aucune marge de décision. La seule décision que vous pouvez prendre est la vitesse à laquelle vous travaillez, et cela ne peut probablement augmenter vos revenus que d'un facteur de deux ou trois.

Un exemple d'emploi avec à la fois de la mesure et de l'effet de levier serait l'acteur principal dans un film. Votre performance peut être mesurée dans le gros du film. Et vous avez un effet de levier dans le sens où votre performance peut le faire ou le briser.

Les PDG ont également à la fois de la mesure et de l'effet de levier. Ils sont mesurés, en ce sens que la performance de l'entreprise est leur performance. Et ils ont un effet de levier en ce sens que leurs décisions font avancer l'ensemble de l'entreprise dans une direction ou une autre.

Je pense que tous ceux qui s'enrichissent par leurs propres efforts se retrouvent dans une situation de mesure et d'effet de levier. Tous ceux à qui je peux penser le font : PDG, stars de cinéma, gestionnaires de fonds spéculatifs, athlètes professionnels. Un bon indice de la présence d'un effet de levier est la possibilité d'échec. Les avantages doivent être compensés par les inconvénients, donc s'il y a un grand potentiel de gain, il doit également y avoir une possibilité terrifiante de perte. Les PDG, les stars, les gestionnaires de fonds et les athlètes

vivent tous avec l'épée au-dessus de leur tête ; au moment où ils commencent à être moins performants, ils sont sortis. Si vous êtes dans un emploi qui se sent en sécurité, vous n'allez pas devenir riche, parce que s'il n'y a pas de danger, il n'y a presque certainement pas d'effet de levier.

Mais vous n'avez pas besoin de devenir un PDG ou une star de cinéma pour être dans une situation de mesure et d'effet de levier. Tout ce que vous avez à faire est de faire partie d'un petit groupe travaillant sur un problème difficile.

Petitesse = Mesure

Si vous ne pouvez pas mesurer la valeur du travail effectué par des employés individuels, vous pouvez vous en approcher. Vous pouvez mesurer la valeur du travail effectué par de petits groupes.

Un niveau auquel vous pouvez mesurer avec précision les revenus générés par les employés est celui de l'ensemble de l'entreprise. Lorsque l'entreprise est petite, vous êtes donc assez proche de mesurer les contributions des employés individuels. Une start-up viable pourrait n'avoir que dix employés, ce qui vous place dans un facteur de dix de mesure de l'effort individuel.

Commencer ou rejoindre une start-up est donc aussi proche que la plupart des gens peuvent dire à son patron, je veux travailler dix fois plus dur, alors s'il vous plaît, payez-moi dix fois plus. Il y a deux différences : vous ne le dites pas à votre patron, mais directement aux clients (pour lesquels votre patron n'est qu'un mandataire après tout), et vous ne le faites pas individuellement, mais avec un petit groupe d'autres personnes ambitieuses.

Il s'agit, normalement, d'un groupe. Sauf dans quelques types de travail inhabituels, comme le jeu d'acteur ou l'écriture de livres, vous ne pouvez pas être la compagnie d'une seule personne. Et les gens avec qui vous travaillez feraient mieux d'être bons, parce que c'est leur travail avec lequel le vôtre sera moyenné.

Une grande entreprise est comme une galère géante conduite par un millier de rameurs. Deux choses maintiennent la vitesse de la cuisine vers le bas. La première est que les rameurs individuels ne voient aucun résultat en

travaillant plus dur. L'autre est que, dans un groupe de milliers de personnes, le rameur moyen est susceptible d'être assez moyen.

Si vous sortez dix personnes au hasard de la grande galère et que vous les mettiez seules dans un bateau, elles pourraient probablement aller plus vite. Ils auraient à la fois de la carotte et du bâton pour les motiver. Un rameur énergique serait encouragé par l'idée qu'il pourrait avoir un effet visible sur la vitesse du bateau. Et si quelqu'un était paresseux, les autres seraient plus susceptibles de le remarquer et de se plaindre.

Mais le véritable avantage du bateau à dix hommes se manifeste lorsque vous sortez les dix meilleurs rameurs de la grande galère et que vous les mettez dans un bateau ensemble. Ils auront toute la motivation supplémentaire qui vient d'être en petit groupe. Mais plus important encore, en sélectionnant ce petit groupe, vous pouvez obtenir les meilleurs rameurs. Chacun sera dans le top 1%. C'est une bien meilleure affaire pour eux de faire la moyenne de leur travail avec un petit groupe de leurs pairs que de faire la moyenne avec tout le monde.

C'est le vrai intérêt des startups. Idéalement, vous vous réunissez avec un groupe d'autres personnes qui veulent aussi travailler beaucoup plus dur et être payées beaucoup plus qu'elles ne le feraient dans une grande entreprise. Et parce que les startups ont tendance à se faire fonder par des groupes d'auto-sélection de personnes ambitieuses qui se connaissent déjà (au moins par réputation), le niveau de mesure est plus précis que celui de la petite taille seule. Une start-up n'est pas seulement dix personnes, mais dix personnes comme vous.

Steve Jobs a dit un jour que le succès ou l'échec d'une start-up dépend des dix premiers employés. Je suis d'accord. Si quoi que ce soit, c'est plus comme les cinq premiers. Être petit n'est pas, en soi, ce qui fait que les startups donnent un coup de pied aux fesses, mais plutôt que les petits groupes peuvent être sélectionnés. Vous ne voulez pas petit dans le sens d'un village, mais petit dans le sens d'une "*all-star team*".

Plus un groupe est grand, plus son membre moyen sera proche de la moyenne de la population dans son ensemble. Donc, toutes les autres choses étant égales, une personne très capable dans une grande entreprise obtient probablement une mauvaise affaire, parce que sa performance est entraînée vers

le bas par la performance globale inférieure des autres. Bien sûr, toutes les autres choses ne sont souvent pas égales : la personne capable peut ne pas se soucier de l'argent, ou peut préférer la stabilité d'une grande entreprise. Mais une personne très capable qui se soucie de l'argent fera généralement mieux de partir et de travailler avec un petit groupe de pairs.

Technologie = Effet de levier

Les startups offrent à quiconque un moyen d'être dans une situation de mesure et d'effet de levier. Ils permettent la mesure parce qu'ils sont petits, et ils offrent un effet de levier parce qu'ils gagnent de l'argent en inventant de nouvelles technologies.

Qu'est-ce que la technologie ? C'est de la *technique*. C'est la façon dont nous faisons tous les choses. Et lorsque vous découvrez une nouvelle façon de faire les choses, sa valeur est multipliée par toutes les personnes qui l'utilisent. C'est la canne à pêche proverbiale, plutôt que le poisson. C'est la différence entre une start-up et un restaurant ou un salon de coiffure. Vous faites frire des œufs ou vous coupez les cheveux un moment à la fois. Alors que si vous résolvez un problème technique dont beaucoup de gens se soucient, vous aidez tous ceux qui utilisent votre solution. C'est un effet de levier.

Si vous regardez l'histoire, il semble que la plupart des gens qui se sont enrichis en créant de la richesse l'ont fait en développant de nouvelles technologies. Vous ne pouvez tout simplement pas faire frire des œufs ou couper les cheveux assez vite. Ce qui a rendu les Florentins riches en 1200, c'est la découverte de nouvelles techniques pour fabriquer le produit de haute technologie de l'époque, le tissu tissé fin. Qu'est-ce qui a rendu les Hollandais riches en 1600 a été la découverte de techniques de construction navale et de navigation qui leur ont permis de dominer les mers de l'Extrême-Orient.

Heureusement, il y a un ajustement naturel entre la petitesse et la résolution de problèmes difficiles. La pointe de la technologie évolue rapidement. La technologie qui a de la valeur aujourd'hui pourrait être sans valeur dans quelques années. Les petites entreprises sont plus à l'origine dans ce monde, parce qu'elles n'ont pas de couches de bureaucratie pour les ralentir. En

outre, les avancées techniques ont tendance à provenir d'approches peu orthodoxes, et les petites entreprises sont moins contraintes par les conventions.

Les grandes entreprises peuvent développer la technologie. Ils ne peuvent tout simplement pas le faire rapidement. Leur taille les rend lents et les empêche de récompenser les employés pour l'effort extraordinaire requis. Ainsi, dans la pratique, les grandes entreprises ne développent des technologies que dans des domaines où de grandes exigences en capital empêchent les start-up de concurrencer avec elles, comme les microprocesseurs, les centrales électriques ou les avions de passagers. Et même dans ces domaines, ils dépendent fortement des startups pour les composants et les idées.

Il est évident que les startups biotechnologiques ou logicielles existent pour résoudre des problèmes techniques difficiles, mais je pense que cela era également vrai dans les entreprises qui ne semblent pas être une question de technologie. McDonald, par exemple, a grandi en concevant un système, la franchise McDonald , qui pouvait ensuite être reproduit à volonté sur toute la surface de la terre. Une franchise McDonald est contrôlée par des règles si précises qu'il s'agit pratiquement d'un logiciel. Écrivez une fois, exécutez partout où. Idem pour WalMart. Sam Walton s'est enrichi non pas en étant un détaillant, mais en concevant un nouveau type de magasin.

Utilisez la difficulté comme guide non seulement pour choisir l'objectif global de votre entreprise, mais aussi aux points de décision en cours de route. Chez Viaweb, l'une de nos règles empiriques a été exécutée à l'étage. Supposons que vous soyez un petit gars agile poursuivi par un gros intimidateur. Vous ouvrez une porte et vous vous retrouvez dans un escalier. Est-ce que tu montes ou descends ? Je dis. L'intimidateur peut probablement descendre aussi vite que vous le pouvez. Monter à l'étage de sa masse sera plus un désavantage. Courir en montant à l'étage est difficile pour vous, mais encore plus difficile pour lui.

Ce que cela signifiait dans la pratique, c'est que nous recherchions délibérément des problèmes difficiles. S'il y avait deux fonctionnalités que nous pourrions ajouter à notre logiciel, toutes deux tout aussi précieuses en proportion de leur difficulté, nous prendrions toujours le plus difficile. Non seulement parce que c'était plus précieux, mais aussi parce que c'était plus difficile. Nous avons été ravis de forcer de grands concurrents plus lents à nous

suivre sur un terrain difficile. Comme les guérilleros, les startups préfèrent le terrain difficile des montagnes, où les troupes du gouvernement central ne peuvent pas les suivre. Je me souviens des moments où nous étions juste épuisés après avoir lutté toute la journée avec un horrible problème technique. Et je serais ravi, parce que quelque chose qui était difficile pour nous serait impossible pour nos concurrents.

Ce n'est pas seulement une bonne façon de gérer une start-up. C'est ce qu'est une start-up. Les investisseurs en capital-risque connaissent ce phénomène et ont une expression pour la décrire : les *barrières à l'entrée*. Si vous allez dans un VC avec une nouvelle idée et que vous lui demandez d'y investir, l'une des premières choses qu'il demandera est, à quel point cela serait-il difficile pour quelqu'un d'autre de se développer ? C'est-à-dire, combien de terrain difficile avez-vous mis entre vous-même et les poursuivants potentiels ? [7] Et vous feriez mieux d'avoir une explication convaincante de la raison pour laquelle votre technologie serait difficile à reproduire. Sinon, dès qu'une grande entreprise en aura connaissance, elle fera la sienne, et avec sa marque, son capital et son influence sur la distribution, elle vous enlèvera votre marché du jour au lendemain. Vous seriez comme des guérilleros pris en plein champ par les forces de l'armée régulière.

Une façon de mettre en place des obstacles à l'entrée est par le biais de brevets. Mais les brevets peuvent ne pas fournir beaucoup de protection. Les concurrents trouvent communément des moyens de contourner un brevet. Et s'ils ne le peuvent pas, ils peuvent simplement le violer et vous inviter à les poursuivre en justice. Une grande entreprise n'a pas peur d'être poursuivie en justice ; c'est une chose quotidienne pour elle. Ils s'assureront que les poursuivre en justice coûte cher et prend beaucoup de temps. Avez-vous déjà entendu parler de Philo Farnsworth ? Il a inventé la télévision. La raison pour laquelle vous n'avez jamais entendu parler de lui, c'est que son entreprise n'était pas celle qui gagnait de l'argent [8]. L'entreprise qui l'a fait était RCA, et la récompense de Farnsworth pour ses efforts a été une décennie de litige en matière de brevets.

Ici, comme souvent, la meilleure défense est une bonne attaque. Si vous pouvez développer une technologie qui est tout simplement trop difficile à dupliquer pour les concurrents, vous n'avez pas besoin d'autres défenses.

Commencez par trouver un problème difficile, puis à chaque point de décision, prenez le choix le plus difficile [9].

Le(s) piège(s)

S'il s'agissait simplement de travailler plus dur qu'un employé ordinaire et d'être payé proportionnellement, ce serait évidemment une bonne affaire de démarrer une start-up. Jusqu'à un moment donné, ce serait plus amusant. Je ne pense pas que beaucoup de gens aiment le rythme lent des grandes entreprises, les réunions interminables, les conversations sur les distributeurs d'eau, les cadres intermédiaires ignorants, et ainsi de suite.

Malheureusement, il y a quelques prises. La première est que vous ne pouvez pas choisir le point de la courbe que vous voulez habiter. Vous ne pouvez pas décider, par exemple, que vous aimeriez travailler deux ou trois fois plus dur et être payé beaucoup plus. Lorsque vous dirigez une start-up, vos concurrents décident à quel point vous travaillez dur. Et ils prennent à peu près tous la même décision : aussi difficile que possible.

L'autre hic, c'est que le gain n'est qu'en moyenne proportionnel à votre productivité. Il y a, comme je l'ai déjà dit, un grand multiplicateur aléatoire dans le succès de toute entreprise. Donc, en pratique, l'accord n'est pas que vous êtes 30 fois plus productif et que vous êtes payé 30 fois plus. C'est que vous êtes 30 fois plus productif et que vous êtes payé entre zéro et mille fois plus. Si la moyenne est de $30x$, la médiane est probablement nulle. La plupart des start-ups disparaissent, et pas seulement les portails de dogfood dont nous avons tous entendu parler pendant l'Internet Bubble. Il est courant pour une start-up de développer un produit vraiment bon, de prendre un peu trop de temps pour le faire, de manquer d'argent et de devoir fermer ses portes.

Une start-up est comme un moustique. Un ours peut absorber un coup et un crabe est blindé contre un, mais un moustique est conçu pour une chose : marquer. Aucune énergie n'est gaspillée sur la défense. La défense des moustiques, en tant qu'espèce, est qu'il y en a beaucoup, mais c'est une petite consolation pour le moustique individuel.

Les startups, comme les moustiques, ont tendance à être une proposition de tout ou rien. Et vous ne savez généralement pas lequel des deux vous allez Pour arriver jusqu'à la dernière minute. Viaweb a failli faire le plein à plusieurs reprises. Notre trajectoire était comme une onde sinusoïdale. Heureusement, nous avons été achetés au sommet du cycle, mais c'était sacrément proche. Alors que nous visitions Yahoo en Californie pour leur parler de la vente de l'assurance, nous avons dû emprunter une salle de conférence pour rassurer un investisseur qui était sur le point de se retirer d'une nouvelle série de financement dont nous avions besoin pour rester en vie.

L'aspect tout ou rien des startups n'était pas quelque chose que nous voulions. Les hackers de Viaweb étaient tous extrêmement réfractaires au risque. S'il y avait eu un moyen de travailler super dur et d'être payé pour cela, sans avoir une loterie mélangée, nous aurions été ravis. Nous aurions beaucoup préféré une chance de 100 % de 1 million de dollars à une chance de 20 % de 10 millions de dollars, même si théoriquement la seconde vaut deux fois plus. Malheureusement, il n'y a actuellement aucun espace dans le monde des affaires où vous pouvez obtenir la première offre.

Le plus proche que vous puissiez obtenir est de vendre votre start-up dans les premiers stades, en renonçant à l'avantage (et au risque) pour un gain plus faible mais garanti. Nous avons eu la chance de le faire, et stupidement, comme nous le pensions alors, nous l'avons laissé passer. Après cela, nous sommes devenus comiquement désireux de vendre. Pour l'année prochaine environ, si quelqu'un exprimait la moindre curiosité à propos de Viaweb, nous essayerions de lui vendre l'entreprise. Mais il n'y avait pas de preneurs, nous avons donc dû continuer.

Cela aurait été une bonne affaire de nous acheter à un stade précoce, mais les entreprises qui font des acquisitions ne sont pas à la recherche de bonnes affaires. Une entreprise assez grande pour acquérir des startups sera assez grande pour être assez conservatrice, et au sein de l'entreprise, les personnes en charge des acquisitions seront parmi les plus conservatrices, car elles sont susceptibles d'être des types d'écoles de commerce qui ont rejoint l'entreprise en retard. Ils préfèrent payer trop cher pour un choix sûr. Il est donc plus facile de vendre une start-up établie, même à une prime importante, qu'une startup à un stade précoce.

Obtenir des utilisateurs

Je pense que c'est une bonne idée de se faire acheter, si vous le pouvez. Diriger une entreprise est différent de la croissance d'une entreprise. C'est tout aussi bien de laisser une grande entreprise prendre le relais une fois que vous atteignez l'altitude de croisière. C'est aussi plus sage, car la vente vous permet de vous diversifier. Que penseriez-vous d'un conseiller financier qui place tous les actifs de ses clients dans une seule action volatile ?

Comment vous faites-vous acheter ? Principalement en faisant les mêmes choses que vous le faisiez si vous n'aviez pas l'intention de vendre l'entreprise. Être rentable, par exemple. Mais se faire acheter est aussi un art à part entière, et que nous avons passé beaucoup de temps à essayer de maîtriser.

Les acheteurs potentiels tarderont toujours s'ils le peuvent. La partie la plus difficile à propos de l'achat est de les amener à agir. Pour la plupart des gens, le facteur de motivation le plus puissant n'est pas l'espoir de gain, mais la peur de la perte. Pour les acquéreurs potentiels, le facteur de motivation le plus puissant est la perspective que l'un de leurs concurrents vous achète. Cela, comme nous l'avons constaté, amène les PDG à avoir les yeux rouges. Le deuxième plus grand est l'inquiétude que, s'ils ne vous achètent pas maintenant, vous continuerez à croître rapidement et vous coûterez plus cher pour acquérir plus tard, ou même devenir un concurrent.

Dans les deux cas, tout se résume aux utilisateurs. Vous pensiez qu'une entreprise sur le point de vous acheter ferait beaucoup de recherches et déciderait par elle-même à quel point votre technologie était précieuse. Pas du tout. Ce qu'ils passent, c'est le nombre d'utilisateurs que vous avez.

En effet, les acquéreurs supposent que les clients savent qui dispose de la meilleure technologie. Et ce n'est pas aussi stupide qu'il n'y paraît. Les utilisateurs sont la seule preuve réelle que vous avez créé de la richesse. La richesse est ce que les gens veulent, et si les gens n'utilisent pas votre logiciel, ce n'est peut-être pas seulement parce que vous êtes mauvais en marketing. C'est peut-être parce que vous n'avez pas fait ce qu'ils veulent.

Les investisseurs en capital-risque ont une liste de signes de danger à surveiller. Près du sommet se trouve l'entreprise dirigée par des techno-weenies qui sont obsédés par la résolution de problèmes techniques intéressants, au lieu de rendre les utilisateurs heureux. Dans une start-up, vous n'essayez pas seulement de résoudre des problèmes. Vous essayez de résoudre les problèmes qui *importent aux utilisateurs*.

Je pense donc que vous devriez faire le test aux utilisateurs, tout comme le font les acquéreurs. Traiter une start-up comme un problème d'optimisation dans lequel la performance est mesurée par le nombre d'utilisateurs. Comme le sait quiconque a essayé d'optimiser le logiciel, la clé est la mesure. Quand vous essayez de deviner où votre programme est lent, et ce qui le rendrait plus rapide, vous devinez presque toujours mal.

Le nombre d'utilisateurs n'est peut-être pas le test parfait, mais il sera très proche. C'est ce dont les acquéreurs se soucient. C'est de cela que dépendent les revenus. C'est ce qui rend les concurrents malheureux. C'est ce qui impressionne les journalistes et les nouveaux utilisateurs potentiels. C'est certainement un meilleur test que vos notions a priori sur les problèmes qu'il est important de résoudre, peu importe à quel point vous êtes techniquement habile.

Entre autres choses, traiter une start-up comme un problème d'optimisation vous aidera à éviter un autre piège dont les investisseurs de capital-risque s'inquiètent, et à juste titre, à prendre beaucoup de temps pour développer un produit. Maintenant, nous pouvons reconnaître cela comme quelque chose que les hackers savent déjà éviter : une optimisation prématuée. Obtenez une version 1.0 dès que vous le pouvez. Jusqu'à ce que vous ayez quelques utilisateurs à mesurer, vous optimisez en fonction des suppositions.

La balle que vous devez surveiller ici est le principe sous-jacent selon lequel la richesse est ce que les gens veulent. Si vous prévoyez de devenir riche en créant de la richesse, vous devez savoir ce que les gens veulent. Si peu d'entreprises font vraiment attention à rendre les clients heureux. À quelle fréquence entrez-vous dans un magasin, ouappelez-vous une entreprise au téléphone, avec un sentiment de peur au fond de votre esprit ? Lorsque vous entendez « votre appel est important pour nous, s'il vous plaît restez en ligne », pensez-vous, oh bien, maintenant tout ira bien ?

Un restaurant peut se permettre de servir occasionnellement un dîner brûlé. Mais dans la technologie, vous cuisinez une chose et c'est ce que tout le monde mange. Ainsi, toute différence entre ce que les gens veulent et ce que vous livrez est multipliée. Vous s'il vous plaît ou ennuyez les clients en gros. Plus vous vous rapprochez de ce qu'ils veulent, plus vous générerez de richesse.

Richesse et pouvoir

Faire de la richesse n'est pas le seul moyen de s'enrichir. Pendant la majeure partie de l'histoire de l'humanité, cela n'a même pas été le plus courant. Jusqu'à il y a quelques années, les principales sources de richesse étaient les mines, les esclaves et les serfs, la terre et le bétail, et les seuls moyens de les acquérir rapidement étaient par l'héritage, le mariage, la conquête ou la confiscation. Naturellement, la richesse avait une mauvaise réputation.

Deux choses ont changé. Le premier était l'État de droit. Pendant la majeure partie de l'histoire du monde, si vous accumuliez d'une manière ou d'une autre une fortune, le souverain ou ses sbires trouveraient un moyen de la voler. Mais en Europe médiévale, quelque chose de nouveau s'est produit. Une nouvelle classe de marchands et de fabricants a commencé à collecter dans les villes¹⁰. Ensemble, ils ont été en mesure de résister au seigneur féodal local. Donc, pour la première fois de notre histoire, les intimidateurs ont cessé de voler l'argent du déjeuner des nerds. C'était naturellement une grande incitation, et peut-être même la principale cause du deuxième grand changement, l'industrialisation.

Beaucoup de choses ont été écrites sur les causes de la révolution industrielle. Mais une condition nécessaire, sinon suffisante, était sûrement que les gens qui ont fait fortune puissent en profiter en paix [11]. Une preuve est ce qui est arrivé aux pays qui ont essayé de revenir à l'ancien modèle, comme l'Union soviétique, et dans une moindre mesure à la Grande-Bretagne sous les gouvernements travaillistes des années 1960 et du début des années 1970. Enlevez l'incitation de la richesse, et l'innovation technique s'arrête.

Rappelez-vous ce qu'est une start-up, économiquement : une façon de dire, je veux travailler plus vite. Au lieu d'accumuler de l'argent lentement en

étant payé un salaire régulier pendant cinquante ans, je veux en finir avec cela dès que possible. Ainsi, les gouvernements qui vous interdisent d'accumuler de la richesse décrètent en effet que vous travaillez lentement. Ils sont prêts à vous laisser gagner 3 millions de dollars sur cinquante ans, mais ils ne sont pas prêts à vous laisser travailler si dur que vous pouvez le faire en deux. Ils sont comme le patron de l'entreprise à qui vous ne pouvez pas aller et dire, je veux travailler dix fois plus dur, alors s'il vous plaît, payez-moi dix fois plus. Sauf que ce n'est pas un patron auquel vous pouvez échapper en créant votre propre entreprise.

Le problème avec le travail lent n'est pas seulement que l'innovation technique se produit lentement. C'est que cela a tendance à ne pas se produire du tout. Ce n'est que lorsque vous êtes délibérément à la recherche de problèmes difficiles, comme moyen d'utiliser la vitesse au plus grand avantage, que vous prenez en place ce genre de projet. Développer de nouvelles technologies est vraiment chiant. C'est, comme Edison l'a dit, "un pour cent d'inspiration et quatre-vingt-dix-neuf pour cent de transpiration". Sans l'incitation de la richesse, personne ne veut le faire. Les ingénieurs travailleront sur des projets sexy comme les avions de chasse et les fusées lunaires pour des salaires ordinaires, mais des technologies plus banales comme les ampoules ou les semi-conducteurs doivent être développées par des entrepreneurs.

Les startups ne sont pas seulement quelque chose qui s'est passé à Silicon Valley au cours des deux dernières décennies. Depuis qu'il est devenu possible de s'enrichir en créant de la richesse, tous ceux qui l'ont fait ont utilisé essentiellement la même recette : la mesure et l'effet de levier, où la mesure provient du travail avec un petit groupe, et l'effet de levier du développement de nouvelles techniques. La recette était la même à Florence en 1200 qu'à Santa Clara aujourd'hui.

Comprendre cela peut aider à répondre à une question importante : pourquoi l'Europe est devenue si puissante. Était-ce quelque chose à propos de la géographie de l'Europe ? Est-ce que les Européens sont en quelque sorte racialement supérieurs ? Était-ce leur religion ? La réponse (ou du moins la cause importante) peut être que les Européens sont montés sur la crête d'une nouvelle idée puissante : permettre à ceux qui ont gagné beaucoup d'argent de la garder.

Une fois que vous êtes autorisé à le faire, les personnes qui veulent s'enrichir peuvent le faire en générant de la richesse au lieu de la voler. La croissance technologique qui en résulte se traduit non seulement par la richesse, mais aussi par la puissance militaire. La théorie qui a conduit à l'avion furtif a été développée par un mathématicien soviétique. Mais parce que l'Union soviétique n'avait pas d'industrie informatique, cela restait pour eux une théorie; ils n'avaient pas de matériel capable d'exécuter les calculs assez rapidement pour concevoir un véritable avion.

À cet égard, la guerre froide enseigne la même leçon que la Seconde Guerre mondiale et, d'ailleurs, la plupart des guerres de l'histoire récente. Ne laissez pas une classe dirigeante de guerriers et de politiciens écraser les entrepreneurs. La même recette qui rend les individus riches rend les pays riches. Laissez les nerds garder leur argent pour le déjeuner, et vous gouvernez le monde.

Chapitre 7

Faites Attention à l'Écart

Quand les gens se soucient suffisamment de quelque chose pour bien le faire, ceux qui le font le mieux ont tendance à être bien meilleurs que tout le monde. Il y a un énorme écart entre Léonard de Vinci et les contemporains de second ordre comme Borgognone. Vous voyez le même écart entre Raymond Chandler et l'écrivain moyen de romans policiers. Un joueur d'échecs professionnel de premier plan pourrait jouer dix mille matchs contre un joueur de club ordinaire sans perdre une seule fois.

Comme les échecs, la peinture ou l'écriture de romans, gagner de l'argent est une compétence très spécialisée. Mais pour une raison quelconque, nous traitons cette compétence de manière différente. Personne ne se plaint quand quelques personnes surpassent tout le reste en jouant aux échecs ou en écrivant des romans, mais quand quelques personnes gagnent plus d'argent que les autres, nous obtenons des éditoriaux qui disent que c'est faux.

Pourquoi ? Le modèle de variation ne semble pas différent de celui de toute autre compétence. Qu'est-ce qui pousse les gens à réagir si fortement lorsque la compétence est de gagner de l'argent ?

Je pense qu'il y a trois raisons pour lesquelles nous traitons le fait de gagner de l'argent comme différent : le modèle trompeur de richesse que nous apprenons en tant qu'enfants ; la manière peu recommandable dont, jusqu'à récemment, la plupart des fortunes étaient accumulées ; et l'inquiétude que de grandes variations de revenus soient quelque peu mauvaises pour la société. Pour autant que je sache, le premier est erroné, le second est obsolète et le troisième empiriquement faux. Se pourrait-il que, dans une démocratie moderne, la variation des revenus soit en fait un signe de santé ?

The Daddy Model of Wealth

Quand j'avais cinq ans, je pensais que l'électricité était créée par des prises électriques. Je ne me suis pas rendu compte qu'il y avait des centrales électriques qui le génèrent. De même, il ne vient pas à l'esprit de la plupart des

enfants que la richesse est quelque chose qui doit être générée. Il semble que ce soit quelque chose qui découle des parents.

En raison des circonstances dans lesquelles ils le rencontrent, les enfants ont tendance à mal comprendre la richesse. Ils la confondent avec de l'argent. Ils pensent qu'il y en a un montant fixe. Et ils la considèrent comme quelque chose qui est distribué par les autorités (et qui devrait donc être distribué également), plutôt que comme quelque chose qui doit être créé (et qui pourrait être créé de manière inégale).

En fait, la richesse n'est pas de l'argent. L'argent n'est qu'un moyen pratique d'échanger une forme de richesse contre une autre. La richesse est la substance sous-jacente - les biens et services que nous achetons. Lorsque vous voyagez dans un pays riche ou pauvre, vous n'avez pas besoin de regarder les comptes bancaires des gens pour savoir dans quel type vous vous en êtes. Vous pouvez *voir* la richesse - dans les bâtiments et les rues, dans les vêtements et la santé des gens.

D'où vient la richesse ? Les gens la créent. C'était plus facile à comprendre lorsque la plupart des gens vivaient dans des fermes et faisaient beaucoup de choses qu'ils voulaient de leurs propres mains. Ensuite, vous pouviez voir dans la maison, les bergeries et le grenier la richesse que chaque famille créait. Il était alors aussi évident que la richesse du monde n'était pas une quantité fixe qui devait être partagée, comme des tranches de tarte. Si vous vouliez plus de richesse, vous pouviez le faire.

C'est tout aussi vrai aujourd'hui, bien que peu d'entre nous créent de la richesse directement pour nous-mêmes (à l'exception de quelques tâches domestiques rudimentaires). La plupart du temps, nous créons de la richesse pour d'autres personnes en échange d'argent, que nous échangeons ensuite contre les formes de richesse que nous voulons [1].

Parce que les enfants sont incapables de créer de la richesse, tout ce qu'ils ont doit leur être donné. Et quand la richesse est quelque chose qu'on vous donne, alors bien sûr, il semble qu'elle devrait être répartie de manière égale [2]. Comme dans la plupart des familles, c'est le cas. Les enfants ont tendance à

faire ça. « Injuste », pleurent-ils, quand un frère ou une sœur reçoit plus qu'un autre.

Dans le monde réel, vous ne pouvez pas continuer à vivre de vos parents. Si vous voulez quelque chose, vous devez soit le faire, soit faire quelque chose de valeur équivalente pour quelqu'un d'autre, afin qu'il vous donne assez d'argent pour l'acheter. Dans le monde réel, la richesse est (à l'exception de quelques spécialistes comme les voleurs et les spéculateurs) quelque chose que vous devez créer, pas quelque chose qui est distribué par papa. Et depuis que la capacité et le désir de le créer varient d'une personne à l'autre, ce n'est pas fait de manière égale.

Vous êtes payé en faisant ou en faisant quelque chose que les gens veulent, et ceux qui gagnent plus d'argent sont souvent tout simplement meilleurs pour faire ce que les gens veulent. Les meilleurs acteurs gagnent beaucoup plus d'argent que les acteurs de la liste B. Les acteurs de la liste B sont peut-être presque aussi charismatiques, mais quand les gens vont au cinéma et regardent la liste des films qui jouent, ils veulent ce coup de force supplémentaire que les grandes stars ont.

Faire ce que les gens veulent n'est pas le seul moyen d'obtenir de l'argent, bien sûr. Vous pouvez également voler des banques, ou solliciter des pots-de-vin ou établir un monopole. De telles astuces expliquent une certaine variation de la richesse, et en fait de certaines des plus grandes fortunes individuelles, mais elles ne sont pas la cause profonde de la variation du revenu. La cause profonde de la variation des revenus, comme l'implique le rasoir d'Occam, est la même que la cause profonde de la variation de toutes les autres compétences humaines.

Aux États-Unis, le PDG d'une grande société publique gagne environ 100 fois plus que la personne moyenne [3]. Les joueurs de basket-ball gagnent environ 128 fois plus, et les joueurs de baseball 72 fois plus. Les éditoriaux citent ce genre de statistiques avec horreur. Mais je n'ai aucun mal à imaginer qu'une personne puisse être 100 fois plus productive qu'une autre. Dans la Rome antique, le prix des esclaves variait d'un facteur de 50 en fonction de leurs compétences [4]. Et c'est sans tenir compte de la motivation, ou de l'effet de

levier supplémentaire en productivité que vous pouvez obtenir de la technologie moderne.

Les éditoriaux sur les salaires des athlètes ou des PDG me rappellent les premiers écrivains chrétiens, qui discutent dès les premiers principes pour savoir si la Terre était ronde, quand ils pouvaient simplement sortir et vérifier [5]. Combien vaut le travail de quelqu'un n'est pas une question de politique. C'est quelque chose que le marché détermine déjà.

« Est-ce qu'ils valent vraiment 100 d'entre nous ? » demandent les éditorialistes. Cela dépend de ce que vous entendez par valeur. Si vous voulez dire valeur dans le sens de ce que les gens paieront pour leurs compétences, la réponse est oui, apparemment.

Les revenus de quelques PDG reflètent une sorte d'acte répréhensible. Mais n'y en a-t-il pas d'autres dont les revenus reflètent vraiment la richesse qu'ils génèrent ? Steve Jobs a sauvé une entreprise qui était en déclin terminal. Et pas seulement comme le fait un spécialiste du redressement, en réduisant les coûts ; il a dû décider quels devraient être les prochains produits d'Apple. Peu d'autres auraient pu le faire. Et quel que soit le cas avec les PDG, il est difficile de voir comment quelqu'un pourrait faire valoir que les salaires des joueurs de basket-ball professionnels ne reflètent pas l'offre et la demande.

Il peut sembler peu probable en principe qu'un individu puisse générer beaucoup plus de richesse qu'un autre. La clé de ce mystère est de revenir sur cette question, valent-ils vraiment 100 d'entre nous ? Une équipe de basket-ball échangerait-elle un de ses joueurs contre 100 personnes au hasard ? À quoi ressemblerait le prochain produit d'Apple si vous remplacez Steve Jobs par un comité de 100 personnes aléatoires ? [6]. Ces choses ne s'adaptent pas de manière linéaire. Peut-être que le PDG ou l'athlète professionnel n'a que dix fois (ce que cela signifie) les compétences et la détermination d'une personne ordinaire. Mais cela fait toute la différence qu'il soit concentré en un seul individu.

Quand nous disons qu'un type de travail est surpayé et un autre sous-payé, que disons-nous vraiment ? Sur un marché libre, les prix sont déterminés par ce que veulent les acheteurs. Les gens aiment le baseball plus

que la poésie, donc les joueurs de baseball font plus d'argent que des poètes. Dire qu'un certain type de travail est sous-payé est donc identique à dire que les gens veulent les mauvaises choses.

Eh bien, bien sûr, les gens veulent de mauvaises choses. Il semble étrange d'être surpris par cela. Et il semble encore plus étrange de dire qu'il est injuste que certains types de travail soient sous-payés [7]. Ensuite, vous dites qu'il est injuste que les gens veulent de mauvaises choses. Il est regrettable que les gens préfèrent la télé-réalité et les corndogs à Shakespeare et aux légumes cuits à la vapeur, mais injuste ? Cela ressemble à dire que le bleu est lourd, ou que le haut est circulaire.

L'apparition du mot "injuste" ici est la signature spectrale indubitable du modèle de papa. Sinon, pourquoi cette idée se produirait-elle dans ce contexte étrange ? Alors que si l'orateur était toujours en train d'opérer sur le modèle papa, et voyait la richesse comme quelque chose qui provenait d'une source commune et devait être partagée, plutôt que quelque chose généré en faisant ce que d'autres personnes voulaient, c'est exactement ce que vous obtiendriez en remarquant que certaines personnes font beaucoup plus de valeur que d'autres.

Lorsque nous parlons de « répartition inégale des revenus », nous devrions également nous demander d'où vient ce revenu ? [8]. Qui a fait la richesse qu'il représente ? Parce que dans la mesure où le revenu varie simplement en fonction de la quantité de richesse que les gens créent, la distribution peut être inégale, mais elle n'est pas injuste.

Le voler

La deuxième raison pour laquelle nous avons tendance à trouver de grandes disparités de richesse alarmantes est que pendant la majeure partie de l'histoire de l'humanité, la façon habituelle de cumuler une fortune était de la voler : dans les sociétés pastorales par des raids de bétail ; dans les sociétés agricoles en s'appropriant les domaines des autres en temps de guerre et en les taxant en temps de paix.

Dans les conflits, ceux qui sont du côté des gagnants recevraient le patrimoine confisqué aux perdants. En Angleterre dans les 1060, lorsque

Guillaume le Conquérant distribuait les domaines des nobles anglo-saxons vaincus à ses disciples, le conflit était militaire. Dans les années 1530, lorsque Henri VIII distribua les domaines des monastères à ses disciples [9], c'était principalement politique. Mais le principe était le même. En effet, le même principe est à l'œuvre maintenant au Zimbabwe.

Dans les sociétés plus organisées, comme la Chine, le dirigeant et ses fonctionnaires ont utilisé la fiscalité au lieu de la confiscation. Mais ici aussi, nous voyons le même principe : la façon de s'enrichir n'était pas de créer de la richesse, mais de servir un dirigeant assez puissant pour s'en appropier.

Cela a commencé à changer en Europe avec la montée de la classe moyenne. Maintenant, nous considérons la classe moyenne comme des gens qui ne sont ni riches ni pauvres, mais à l'origine, ils étaient un groupe distinct. Dans une société féodale, il n'y a que deux classes : une aristocratie guerrière et les serfs qui travaillent leurs domaines. La classe moyenne était un nouveau, troisième groupe qui vivait dans les villes et subvenait à leurs besoins par la fabrication et le commerce.

À partir des dixième et onzième siècles, les petits nobles et les anciens serfs se sont regroupés dans des villes qui sont progressivement devenues assez puissantes pour ignorer le seigneur féodal local [10]. Comme les serfs, la classe du milieu de la vie en grande partie en créant de la richesse. (Dans les villes portuaires comme Gênes et Pise, ils se sont également engagés dans la piraterie.) Mais contrairement aux serfs, ils étaient incités à en créer beaucoup. Toute richesse créée par un serf appartenait à son maître. Il n'y avait pas beaucoup d'intérêt à faire plus que ce que vous pouviez cacher. Alors que l'indépendance des citadins leur a permis de conserver toute richesse qu'ils créaient.

Une fois qu'il est devenu possible de s'enrichir en créant de la richesse, la société dans son ensemble a commencé à s'enrichir très rapidement. Presque tout ce que nous avons a été créé par la classe moyenne. En effet, les deux autres classes ont effectivement disparu dans les sociétés industrielles, et leurs noms ont été donnés aux deux extrémités de la classe moyenne. (Dans le sens original du terme, Bill Gates est de la classe moyenne.)

Mais ce n'est qu'à la révolution industrielle que la création de richesse a définitivement remplacé la corruption comme meilleur moyen de s'enrichir. En Angleterre, du moins, la corruption n'est devenue démodée (et en fait n'a commencé à être appelée "corruption") que lorsqu'il a commencé à y avoir d'autres moyens plus rapides de devenir riche.

L'Angleterre du XVIIe siècle ressemblait beaucoup au tiers monde d'aujourd'hui, dans ce bureau du gouvernement était une voie reconnue vers la richesse. Les grandes fortunes de cette époque dérivaient encore plus de ce que nous appellerions aujourd'hui la corruption que du commerce [11]. Au XIXe siècle, cela avait changé. Il y a continué à y avoir des pots-de-vin, comme il y en a encore partout, mais la politique avait alors été laissée à des hommes qui étaient plus motivés par la vanité que par la cupidité. La technologie avait permis de créer de la richesse plus rapidement que vous ne pouviez la voler. Le riche prototypique du XIXe siècle n'était pas un courtisan, mais un industriel.

Avec la montée de la classe moyenne, la richesse a cessé d'être un jeu à somme nulle. Jobs et Wozniak n'ont pas eu à nous rendre pauvres pour se rendre riches. Bien au contraire : ils ont créé des choses qui ont rendu nos vies matériellement plus riches. Ils devaient le faire, ou nous n'aurions pas payé pour eux.

Mais comme pendant la majeure partie de l'histoire du monde, la principale voie vers la richesse était de la voler, nous avons tendance à nous méfier des riches. Les étudiants de premier cycle idéalistes trouvent le modèle de richesse de leur enfant inconsciemment préservé confirmé par d'éminents écrivains du passé. C'est le cas de l'erreur de rencontrer le dépassé..

« Derrière chaque grande fortune, il y a un crime », a écrit Balzac. Sauf qu'il ne l'a pas fait. Ce qu'il a réellement dit, c'est qu'une grande fortune sans cause apparente était probablement due à un crime suffisamment bien exécuté pour qu'il ait été oublié. Si nous parlions de l'Europe en 1000, ou de la majeure partie du tiers monde aujourd'hui, la citation erronée standard serait exacte. Mais Balzac a vécu dans la France du XIXe siècle, où la révolution industrielle était bien avancée. Il savait que vous pouviez faire fortune sans la voler. Après tout, il l'a fait lui-même, en tant que romancier populaire [12].

Seuls quelques pays (sans coïncidence, les plus riches) ont atteint ce stade. Dans la plupart des cas, la corruption a toujours le dessus. Dans la plupart des cas, le moyen le plus rapide d'obtenir de la richesse est de la voler. Et donc, lorsque nous voyons des différences de revenus croissantes dans un pays riche, il y a une tendance à craindre qu'il ne redevienne un autre Venezuela. Je pense que c'est le contraire qui se produit. Je pense que vous voyez un pays avoir une longueur d'avance sur le Venezuela.

Le levier de la technologie

La technologie augmentera-t-elle l'écart entre les riches et les pauvres ? Cela augmentera certainement l'écart entre le productif et le non productif. C'est tout l'intérêt de la technologie. Avec un tracteur, un agriculteur énergique pouvait labourer six fois plus de terres en une journée qu'avec une équipe de chevaux. Mais seulement s'il maîtrisait un nouveau type d'agriculture.

J'ai vu le levier de la technologie croître visiblement à mon époque. Au lycée, je gagnais de l'argent en tondant les pelouses et en ramassant de la crème glacée chez Baskin-Robbins. C'était le seul type de travail disponible à l'époque. Maintenant, les lycéens pouvaient écrire des logiciels ou concevoir des sites Web. Mais seuls certains d'entre eux le feront ; les autres ramasseront toujours de la crème glacée.

Je me souviens très bien quand, en 1985, l'amélioration de la technologie m'a permis d'acheter mon propre ordinateur. En quelques mois, je l'ai utilisé pour gagner de l'argent en tant que programmeur indépendant. Quelques années auparavant, je n'aurais pas pu le faire. Quelques années auparavant, il n'y avait pas de programmeur indépendant. Mais Apple a créé de la richesse, sous la forme d'ordinateurs puissants et peu coûteux, et les programmeurs se sont immédiatement mis au travail en l'utilisant pour en créer plus.

Comme le suggère cet exemple, la vitesse à laquelle la technologie augmente notre capacité de production est probablement polynomiale, plutôt que linéaire. Nous devrions donc nous attendre à voir une variation toujours croissante de la productivité individuelle au fil du temps. Cela augmentera-t-il l'écart entre les riches et les pauvres ? Cela dépend de l'écart que vous voulez dire. La technologie devrait augmenter l'écart de revenu, mais elle semble

diminuer d'autres écarts. Il y a cent ans, les riches menaient une vie différente de celle des gens ordinaires. Ils vivaient dans des maisons pleines de serviteurs, portaient des vêtements élaborés et inconfortables et voyageaient dans des voitures tirées par des équipes de chevaux qui avaient eux-mêmes besoin de leurs propres maisons et de leurs propres serviteurs. Maintenant, grâce à la technologie, les riches vivent plus comme la personne moyenne.

Les voitures sont un bon exemple de pourquoi. Il est possible d'acheter des voitures chères faites à la main qui coûtent des centaines de milliers de dollars. Mais il n'y a pas grand intérêt. Les entreprises gagnent plus d'argent en construisant un grand nombre de voitures ordinaires qu'un petit nombre de voitures chères. Ainsi, une entreprise qui fabrique une voiture produite en série peut se permettre de dépenser beaucoup plus pour sa conception. Si vous achetez une voiture sur mesure, quelque chose se cassera toujours. Le seul but d'en acheter un maintenant est d'annoncer que vous le pouvez.

Ou envisagez des montres. Il y a cinquante ans, en dépensant beaucoup d'argent pour une montre, vous pouviez obtenir de meilleures performances. Lorsque les montres avaient des mouvements mécaniques, les montres chères gardaient un meilleur temps. Pas plus. Depuis l'invention du mouvement à quartz, un Timex ordinaire est plus précis qu'un Patek Philippe qui coûte des centaines de milliers de dollars [13]. En effet, comme avec des voitures chères, si vous êtes déterminé à dépenser beaucoup d'argent pour une montre, vous devez supporter certains inconvénients pour le faire : en plus d'afficher un temps erroné, les montres mécaniques doivent être remontées.

La seule chose que la technologie ne peut pas dévaloriser, c'est la marque. C'est précisément la raison pour laquelle nous en entendons de plus en plus parler. La marque est le résidu laissé à mesure que les différences substantielles entre les riches et les pauvres s'évaporent. Mais l'étiquette que vous avez sur vos affaires est beaucoup plus petite que de l'avoir plutôt que de ne pas l'avoir. En 1900, si vous gardiez une voiture, personne ne demandait quelle année ou quelle marque c'était. Si vous en aviez une, vous étiez riche. Et si vous n'étiez pas riche, vous preniez l'omnibus ou marchiez. Maintenant, même les Américains les plus pauvres conduisent des voitures, et ce n'est que parce que nous sommes si bien formés par la publicité que nous pouvons même reconnaître les voitures particulièrement chères [14].

Le même schéma s'est manifesté dans l'industrie après l'industrie. S'il y a suffisamment de demande pour quelque chose, la technologie le rendra assez bon marché pour être vendu en gros volumes [15], et les versions produites en série seront, sinon meilleures, au moins plus pratiques. Et il n'y a rien de plus que la commodité. Les gens riches que je connais conduisent les mêmes voitures, portent les mêmes vêtements, ont le même type de meubles et mangent les mêmes aliments que mes autres amis. Leurs maisons sont dans des quartiers différents, ou si dans le même quartier sont de tailles différentes, mais en eux, la vie est similaire. Les maisons sont fabriquées en utilisant les mêmes techniques de construction et contiennent à peu près les mêmes objets. C'est gênant de faire quelque chose de cher et de personnalisé.

Les riches passent leur temps comme tout le monde. Bertie Wooster semble avoir disparu depuis longtemps. Maintenant, la plupart des gens qui sont assez riches pour ne pas travailler le font de toute façon. Ce n'est pas seulement la pression sociale qui les rend ; l'oisiveté est solitaire et démoralisante.

Nous n'avons pas non plus les distinctions sociales qu'il y avait il y a cent ans. Les romans et les manuels d'étiquette de cette période se lisent maintenant comme des descriptions d'une étrange société tribale. « En ce qui concerne le maintien des amitiés... » laisse entendre le *Book of Household Management* (1880) de Mme Beeton, « il peut s'avérer nécessaire, dans certains cas, pour une maîtresse d'abandonner, en assumant la responsabilité d'un ménage, beaucoup de ceux qui ont commencé dans la première partie de sa vie. » On s'attendait à ce qu'une femme qui a épousé un homme riche laisse tomber des amis qui ne l'ont pas fait. Vous sembleriez barbare si vous vous comportiez de cette façon aujourd'hui. Vous auriez aussi une vie très ennuyeuse. Les gens ont encore tendance à se séparer quelque peu, mais beaucoup plus sur la base de l'éducation que sur la richesse [16].

Matériellement et socialement, la technologie semble réduire l'écart entre les riches et les pauvres, et non l'augmenter. Si Lénine se promenait dans les bureaux d'une entreprise comme Yahoo ou Intel ou Cisco, il penserait que le communisme a gagné. Tout le monde porterait les mêmes vêtements, aurait le même type de bureau (ou plutôt, cubique) avec le même mobilier, et s'adresserait les uns aux autres par leurs prénoms plutôt que par des

honorifiques. Tout semblerait exactement comme il l'avait prédit, jusqu'à ce qu'il regarde leurs comptes bancaires. Oups.

Est-ce un problème si la technologie augmente cet écart ? Cela ne semble pas être si loin. Au fur et à mesure qu'il augmente l'écart de revenu, il semble diminuer la plupart des autres écarts.

Alternative à un Axiome

On entend souvent une politique critiquée au motif qu'elle augmenterait l'écart de revenu entre les riches et les pauvres. Comme si c'était un postulat comme quoi ce serait mauvais. Il est peut-être vrai que l'augmentation de la variation des revenus serait mauvaise, mais je ne vois pas comment nous pouvons dire que c'est *axiomatique*.

En effet, cela peut même être faux, dans les démocraties industrielles. Dans une société de serfs et de seigneurs de guerre, certainement, la variation des revenus est le signe d'un problème sous-jacent. Mais le servage n'est pas la seule cause de variation des revenus. Un pilote de 747 ne gagne pas 40 fois plus qu'un commis à la caisse parce qu'il est un seigneur de guerre qui la tient d'une manière ou d'une autre. Ses compétences sont tout simplement beaucoup plus précieuses.

J'aimerais proposer une idée alternative : dans une société moderne, la variation croissante des revenus est un signe de santé. La technologie semble augmenter la variation de la productivité plus rapidement que les taux linéaires. Si nous ne voyons pas de variation correspondante dans le revenu, il y a trois explications possibles : (a) que l'innovation technique a cessé, (b) que les personnes qui créeraient le plus de richesse ne le font pas, ou (c) qu'elles ne sont pas payées pour cela.

Je pense que nous pouvons dire en toute sécurité que (a) et (b) seraient mauvais. Si vous n'êtes pas d'accord, essayez de vivre pendant un an en utilisant uniquement les ressources disponibles pour le noble franc moyen en 800, et faites-nous rapport. (Je serai généreux et je ne vous renverrai pas à l'âge de pierre.)

La seule option, si vous allez avoir une société de plus en plus prospère sans variation croissante des revenus, semble être la (c), que les gens créeront beaucoup de richesse sans être payés pour cela. Que Jobs et Wozniak, par exemple, travailleront joyeusement 20 heures par jour pour produire l'ordinateur Apple pour une société qui leur permet, après impôts, de garder juste assez de leurs revenus pour correspondre à ce qu'ils auraient fait en travaillant 9 à 5 dans une grande entreprise.

Les gens créeront-ils de la richesse s'ils ne peuvent pas être payés pour cela ? Seulement si c'est amusant. Les gens écriront des systèmes d'exploitation gratuitement. Mais ils ne les installeront pas, ne répondront pas aux appels d'assistance et ne formeront pas les clients à les utiliser. Et au moins 90 % du travail que font même les plus grandes entreprises technologiques est de ce deuxième type, non édifiant.

Tous les types peu amusants de création de richesse ralentissent considérablement dans une société qui confisque les fortunes privées. Nous pouvons le confirmer de manière empirique. Supposons que vous entendiez un bruit étrange qui, selon vous, pourrait être dû à un ventilateur à proximité. Vous éteignez le ventilateur et le bruit s'arrête. Vous rallumez le ventilateur et le bruit recommence. Éteint, calme. Oh, du bruit. En l'absence d'autres informations, il semblerait que le bruit soit causé par le ventilateur.

À différents moments et à divers endroits de l'histoire, la question de savoir si vous pouviez cumuler une fortune en créant de la richesse a été activée et désactivée. Le nord de l'Italie en 800, on (les seigneurs de guerre le voleraient). Le nord de l'Italie en 1100, on. Le Centre de la France en 1100, au large (toujours féodal), on. L'Angleterre en 1800, on. L'Angleterre en 1974, off (taxe de 98 % sur les revenus de placement). États-Unis en 1974, on. Nous avons même eu une étude jumelle : l'Allemagne de l'Ouest, on ; l'Allemagne de l'Est, off. Dans tous les cas, la création de richesse semble apparaître et disparaître comme le bruit d'un ventilateur au fur et à mesure que vous allumez et éteignez la perspective de la garder.

Il y a un certain élan en jeu. Il faut probablement au moins une génération pour transformer les gens en Allemands de l'Est (heureusement pour l'Angleterre). Mais s'il s'agissait simplement d'un ventilateur que nous étudions,

sans tout le bagage supplémentaire qui provient du sujet controversé de la richesse, personne n'aurait le doute que le ventilateur était à l'origine du bruit.

Si vous supprimez les variations de revenus, que ce soit en volant des fortunes privées, comme le faisaient les dirigeants féodaux, ou en les taxant, comme l'ont fait certains gouvernements modernes, le résultat semble toujours être le même. La société dans son ensemble finit par être plus pauvre.

Si j'avais le choix de vivre dans une société où j'étais matériellement beaucoup mieux que maintenant, mais où j'étais parmi les plus pauvres, ou dans une société où j'étais le plus riche, mais beaucoup moins bien que maintenant, je prendrais la première option. Si j'avais des enfants, il serait sans doute immoral de ne pas le faire. C'est la pauvreté absolue que vous voulez éviter, pas la pauvreté relative. Si, comme le montrent jusqu'à présent, vous devez avoir l'un ou l'autre dans votre société, prenez la pauvreté relative.

Vous avez besoin de gens riches dans votre société, non pas tant parce qu'en dépensant leur argent, ils créent des emplois, mais à cause de ce qu'ils doivent faire pour devenir riche. Je ne parle pas de l'effet de ruissellement ici. Je ne dis pas que si vous laissez Henry Ford s'enrichir, il vous embauchera comme serveur à sa prochaine fête. Je dis qu'il vous fera un tracteur pour remplacer votre cheval.

Chapitre 8

Un Plan Pour le Spam

Je pense qu'il est possible d'arrêter le spam, et que les filtres basés sur le contenu sont le moyen d'y parvenir. Le talon d'Achille des spameurs est leur message. Ils peuvent contourner toute autre barrière que vous mettez en place. Ils l'ont fait jusqu'à présent, du moins. Mais ils doivent transmettre leur message, quel qu'il soit. Si nous pouvons écrire un logiciel qui reconnaît leurs messages, il n'y a aucun moyen qu'ils puissent contourner cela [1].

Pour le destinataire, le spam est facilement reconnaissable. Si vous embauchiez quelqu'un pour lire votre courrier et se débarasser du spam, il aurait peu de mal à le faire. Combien devons-nous faire, à l'absence d'IA, pour automatiser ce processus ?

Je pense que nous serons en mesure de résoudre le problème avec des algorithmes assez simples. En fait, j'ai constaté que vous pouvez filtrer le spam d'aujourd'hui de manière acceptable en n'utilisant rien de plus qu'une combinaison bayésienne des probabilités de spam des mots individuels. En utilisant un filtre bayésien légèrement modifié (comme décrit ci-dessous), nous manquons maintenant moins de 5 pour 1000 spams, avec 0 faux positifs.

L'approche statistique n'est généralement pas la première que les gens essaient lorsqu'ils écrivent des filtres anti-spam. Le premier instinct de la plupart des hackers est d'essayer d'écrire un logiciel qui reconnaît les propriétés individuelles du spam. Vous regardez les spams et vous pensez, le culot de ces gars d'essayer de m'envoyer du courrier qui commence par "Cher ami" ou qui a une ligne d'objet qui est toute en majuscules et se termine par huit points d'exclamation. Je peux filtrer ces choses avec environ une ligne de code.

Et c'est ce que vous faites, et au début, cela fonctionne. Quelques règles simples enlèveront une grosse partie de votre spam entrant. Le fait de chercher le mot clic attrapera 79,7 % des e-mails dans mon corpus de spam, avec seulement 1,2 % de faux positifs.

J'ai passé environ six mois à écrire des logiciels qui cherchaient des fonctionnalités de spam individuelles avant d'essayer l'approche statistique. Ce

que j'ai trouvé, c'est que reconnaître que les derniers pourcentages des spams sont devenus très difficiles, et qu'au fur et à mesure que je rendais les filtres plus stricts, j'ai obtenu plus de faux positifs.

Les faux positifs sont des e-mails innocents qui sont identifiés à tort comme des spams. Pour la plupart des utilisateurs, le manque d'e-mails légitimes est d'un ordre de grandeur pire que la réception de spam, de sorte qu'un filtre qui donne de faux positifs est comme un remède contre l'acné qui comporte un risque de mort pour le patient.

Plus un utilisateur reçoit de spam, moins il est susceptible de remarquer un courrier innocent dans son dossier spam. Et étrangement, plus vos filtres anti-spam sont bons, plus les faux positifs deviennent dangereux, car lorsque les filtres sont vraiment bons, les utilisateurs seront plus susceptibles d'ignorer tout ce qu'ils attrapent.

Je ne sais pas pourquoi j'ai évité d'essayer l'approche statistique pendant si longtemps. Je pense que c'est parce que je suis devenu accro à essayer d'identifier moi-même les fonctionnalités de spam, comme si je jouais à une sorte de jeu compétitif avec les spameurs. (Les non-hackers ne s'en rendent pas souvent compte, mais la plupart des hackers sont très compétitifs.) Lorsque j'ai essayé l'analyse statistique, j'ai immédiatement constaté qu'elle était beaucoup plus intelligente que je ne l'avais été. Il a découvert, bien sûr, que des termes comme *virtumundo* et adolescents étaient de bons indicateurs de spam. Mais il a également découvert que per et FL et ff0000 sont de bons indicateurs de spam. En fait, ff0000 (HTML pour rouge vif) s'avère être un aussi bon indicateur de spam que n'importe quel terme pornographique.

Voici un croquis de la façon dont je fais le filtrage statistique. Je commence par un corpus de spam et un corpus de courriers non spam. Pour le moment, chacun contient environ 4000 messages. Je scanne l'ensemble du texte, y compris les en-têtes et le HTML et Javascript intégrés, de chaque message dans chaque corpus. Je considère actuellement les caractères alphanumériques, les tirets, les apostrophes et les signes de dollar comme faisant partie des jetons, et tout le reste comme un séparateur de jetons. (Il y a probablement place à l'amélioration ici.) J'ignore les jetons qui sont tous des chiffres, et j'ignore

également les commentaires HTML, sans même les considérer comme des séparateurs de jetons.

Je compte le nombre de fois où chaque jeton (en ignorant le cas, actuellement) se produit dans chaque corpus. À ce stade, je me retrouve avec deux grandes tables de hachage, une pour chaque corpus, mappant les jetons au nombre d'occurrences.

Ensuite, je crée une troisième table de hachage, cette fois en cartographiant chaque jeton à la probabilité qu'un e-mail le contenant soit un spam, $P_{spam|w}$ que je calcule comme suit :

$$r_g = \min(1, 2(good(w)/G)), \quad r_b = \min(1, bad(w)/B)$$

$$P_{spam|w} = \max(.01, \min(.99, r_b/(r_g + r_b)))$$

Où w est le jeton dont nous calculons la probabilité, *bon* et *mauvais* sont les tables de hachage que j'ai créées dans la première étape, et G et B sont le nombre de messages non-spam et de spam respectivement.

Je veux biaiser légèrement les probabilités pour éviter les faux positifs, et par essais et erreurs, j'ai constaté qu'une bonne façon de le faire est de doubler tous les chiffres en *bon état*. Cela permet de faire la distinction entre les mots qui apparaissent occasionnellement dans les e-mails légitimes et les mots qui ne le font presque jamais. Je ne considère que les mots qui se produisent plus de cinq fois au total (en fait, en raison du doublement, se produire trois fois dans le courrier non spam serait suffisant). Et puis il y a la question de la probabilité à attribuer aux mots qui se trouvent dans un corpus mais pas dans l'autre. Encore une fois, par essais et erreurs, j'ai choisi .01 et .99. Il y a peut-être de la place pour l'accord ici, mais au fur et à mesure que le corpus se développe, un tel accord se produira automatiquement de toute façon.

Le plus observateur remarquera que bien, que je considère chaque corpus comme un seul long flux de texte à des fins de comptage des occurrences, j'utilise le nombre d'e-mails dans chacun, plutôt que leur longueur combinée, comme diviseur dans le calcul des probabilités de spam. Cela ajoute un autre léger biais pour se protéger contre les faux positifs.

Lorsque le nouveau courrier arrive, il est scanné en jetons, et les quinze jetons les plus intéressants, ou l'intérêt est mesuré en fonction de la distance qui sépare leur probabilité de spam d'une probabilité neutre. 5, sont utilisés pour calculer la probabilité que le courrier soit du spam. Si w_1, \dots, w_{15} sont les quinze jetons les plus intéressants, vous calculez la probabilité combinée de cette manière :

$$P_{spam} = \frac{\prod_{i=1}^{15} P_{spam|w_i}}{\prod_{i=1}^{15} P_{spam|w_i} + \prod_{i=1}^{15} (1 - P_{spam|w_i})}$$

Une question qui se pose dans la pratique est la probabilité d'attribuer à un mot que vous n'avez jamais vu, c'est-à-dire un mot qui n'apparaît pas dans la table de hachage des probabilités de mots. J'ai constaté, encore une fois par essais et erreurs, que .4 est un bon nombre à utiliser. Si vous n'avez jamais vu un mot auparavant, il est probablement assez innocent ; les mots de spam ont tendance à être trop familiers.

Je traite le courrier comme du spam si l'algorithme ci-dessus lui donne une probabilité de plus de .9 d'être du spam. Mais dans la pratique, peu importe où je mets ce seuil, car peu de probabilités finissent au milieu de la fourchette.

Un grand avantage de l'approche statistique est que vous n'avez pas besoin de lire autant de spams. Au cours des six derniers mois, j'ai lu littéralement des milliers de spams, et c'est vraiment un peu démoralisant. Norbert Wiener a dit que si vous êtes en compétition avec des esclaves, vous devenez un esclave, et il y a quelque chose de tout aussi dégradant à rivaliser avec les spameurs. Pour reconnaître les caractéristiques individuelles du spam, vous devez essayer d'entrer dans l'esprit du spameur, et franchement, je veux passer le moins de temps possible dans l'esprit des spameurs.

Mais le véritable avantage de l'approche bayésienne, bien sûr, est que vous savez ce que vous mesurez. Les filtres de reconnaissance des fonctionnalités comme SpamAssassin attribuent un « score » de spam à l'e-mail. L'approche bayésienne attribue une probabilité réelle. Le problème avec un « score » est que personne ne sait ce que cela signifie. L'utilisateur ne sait pas ce que cela signifie, mais pire encore, le développeur du filtre non plus. Combien

de points un e-mail devrait-il recevoir pour avoir le mot sexe dedans ? Une probabilité peut bien sûr être erronée, mais il y a peu d'ambiguïté sur ce qu'elle signifie, ou sur la façon dont les preuves doivent être combinées pour les calculer. Sur la base de mon corpus, *sexe* indique une probabilité de .97 que l'e-mail contenant soit un spam, tandis que *sexy* indique .99 de probabilité. Et la règle de Bayes, tout aussi sans ambiguïté, dit qu'un e-mail contenant les deux mots, en l'absence (peu probable) de toute autre preuve, aurait une chance de 99,97 % d'être un spam.

Parce qu'elle mesure les probabilités, l'approche bayésienne prend en compte toutes les preuves dans l'e-mail, bonnes et mauvaises. Les mots qui se produisent disproportionnellement rarement dans le spam (comme *bien que* ou *ce soir* ou *apparemment*) contribuent autant à diminuer la capacité du problème que les mauvais mots comme le *désabonnement* et *l'adhésion* le font pour l'augmenter. Ainsi, un e-mail autrement innocent qui inclut le mot sexe ne sera pas étiqueté comme spam.

Idéalement, bien sûr, les probabilités devraient être calculées de manière individuelle pour chaque utilisateur. Je reçois beaucoup d'e-mails contenant le mot Lisp, et (jusqu'à présent) aucun spam ne le fait. Donc, un mot comme ça est en fait une sorte de mot de passe pour m'envoyer du courrier. Dans mon précédent logiciel de filtrage du spam, l'utilisateur pouvait configurer une liste de ces mots et le courrier les contenant dépasserait automatiquement les filtres. Sur ma liste, j'ai mis des mots comme Lisp et aussi mon code postal, afin que les reçus (sinon plutôt spammeux) des commandes en ligne passent. Je pensais être très intelligent, mais j'ai constaté que le filtre bayésien faisait la même chose pour moi, et j'ai de plus découvert beaucoup de mots auxquels je n'avais pas pensé.

Quand j'ai dit au début que nos filtres laissent passer moins de 5 spams pour 1000 avec 0 faux positifs, je parle de filtrer mon courrier sur la base d'un corpus de mon courrier. Mais ces chiffres ne sont pas trompeurs, car c'est l'approche que je préconise : filtrer le courrier de chaque utilisateur en fonction du spam et du courrier non-spam qu'il reçoit. Essentiellement, chaque utilisateur devrait avoir deux boutons de suppression, de suppression ordinaire et de suppression en tant que spam. Tout ce qui est supprimé en tant que spam va dans le corpus de spam, et tout le reste va dans le corpus de non-spam.

Vous pourriez démarrer les utilisateurs avec un filtre de semences, mais en fin de compte, chaque utilisateur devrait avoir ses propres probabilités par mot en fonction du courrier réel qu'il reçoit. Cela (a) rend les filtres plus efficaces, (b) permet à chaque utilisateur de décider de sa propre définition précise du spam, et (c) peut-être le meilleur de tous, rend difficile pour les spameurs d'accorder les mails pour passer à travers les filtres. Si une grande partie du cerveau du filtre se trouve dans les bases de données individuelles, alors il suffit de régler les spams pour passer à travers les filtres à semences ne garantissent rien sur la façon dont ils passeront à travers les filtres variés et beaucoup plus entraînés des utilisateurs individuels.

Le filtrage du spam basé sur le contenu est souvent combiné à une liste d'expéditeurs dont le courrier peut être accepté sans filtrage. Un moyen facile de créer une telle liste blanche est de conserver une liste de toutes les adresses auxquelles l'utilisateur a déjà envoyé du courrier. Si un lecteur de courrier a un bouton de suppression en tant que spam, vous pouvez également ajouter l'adresse de chaque email que l'utilisateur a supprimé en tant que corbeille ordinaire.

Je suis un défenseur des listes blanches, mais plus comme un moyen d'économiser le calcul, que comme un moyen d'améliorer le filtrage. J'avais l'habitude de penser que les listes blanches faciliteraient le filtrage, parce que vous n'auriez qu'à filtrer les e-mails de personnes dont vous n'aviez jamais entendu parler, et quelqu'un qui vous envoie du courrier pour la première fois est contraint par convention dans ce qu'il peut vous dire. Quelqu'un que vous connaissez déjà pourrait vous envoyer un e-mail parlant de sexe, mais quelqu'un qui vous envoie du courrier pour la première fois ne serait pas susceptible de le faire. Le problème est que les gens peuvent avoir plus d'une adresse e-mail, donc une nouvelle adresse ne garantit pas que l'expéditeur vous écrit pour la première fois. Il n'est pas rare qu'un vieil ami (surtout s'il s'agit d'un hacker) vous envoie soudainement un e-mail avec une nouvelle adresse, de sorte que vous ne pouvez pas risquer de faux positifs en filtrant le courrier à partir d'adresses inconnues de manière particulièrement stricte.

Dans un sens, cependant, mes filtres incarnent eux-mêmes une sorte de liste blanche (et de liste noire) parce qu'ils sont basés sur des messages entiers, y compris les en-têtes. Donc, dans cette mesure, ils « connaissent » les adresses

email des expéditeurs de confiance et même les itinéraires par lesquels le courrier me parvient. Et ils savent la même chose du spam, y compris les noms des serveurs, les versions de courrier et les protocoles.

Si je pensais pouvoir maintenir les taux actuels de filtrage du spam, je considérerais ce problème comme résolu. Mais cela ne signifie pas grand-chose de pouvoir filtrer la plupart des spams actuels, car le spam évolue. En effet, la plupart des techniques antispam jusqu'à présent ont été comme des pesticides qui ne font rien d'autre que créer une nouvelle souche résistante d'insectes.

J'ai plus d'espoir sur les filtres bayésiens, car ils évoluent avec le spam. Ainsi, alors que les spameurs commencent à utiliser *viagra* au lieu de *viagra* pour échapper aux filtres anti-spam simples d'esprit basés sur des mots individuels, les filtres bayésiens le remarquent automatiquement. En effet, *viagra* est une preuve beaucoup plus accablante que le viagra, et les filtres bayésiens savent exactement à quel point.

Pourtant, quiconque propose un plan de filtrage du spam doit être en mesure de répondre à la question : si les spameurs savaient exactement ce que vous faisiez, dans quelle mesure pourraient-ils vous dépasser ? Par exemple, je pense que si le filtrage du spam basé sur la somme de contrôle devient un obstacle sérieux, les spameurs passeront simplement aux techniques mad-lib pour générer des corps de messages.

Pour battre les filtres bayésiens, il ne suffirait pas aux spameurs de rendre leurs e-mails uniques ou d'arrêter d'utiliser des mots coquins individuels. Ils auraient à faire en sorte que leurs courriers ne se distinguent pas de votre courrier ordinaire. Et je pense que cela les limiterait gravement. Le spam est principalement des argumentaires de vente, donc à moins que votre courrier ordinaire ne soit entièrement un argumentaire de vente, les spams auront inévitablement un caractère différent. Et les spameurs devraient aussi, bien sûr, changer (et continuer à changer) toute leur infrastructure, parce que sinon les en-têtes auraient l'air aussi mauvais pour les filtres bayésiens que jamais, peu importe ce qu'ils ont fait au corps du message. Je n'en sais pas assez sur l'infrastructure utilisée par les spameurs pour savoir à quel point il serait difficile de faire paraître les en-têtes innocents, mais je suppose que ce serait encore plus difficile que de rendre le message innocent.

En supposant qu'ils puissent résoudre le problème des en-têtes, le spam du futur ressemblera probablement à ceci :

***"SALUT. DÉCOUVREZ CE QUI SUIT:
HTTP://WWW.27MEG.COM/FOO"***

Parce qu'il s'agit à peu près autant de pitch de vente que le filtrage basé sur le contenu quittera la salle du spameur à faire. (En effet, il sera même difficile d'obtenir ces filtres passés, car si tout le reste de l'e-mail est neutre, la probabilité de spam dépendra de l'URL, et il faudra un certain effort pour que cela ait l'air neutre.)

Les spameurs vont des entreprises qui gèrent des soi-disant listes d'adhésion qui n'essaient même pas de dissimuler leur identité, aux gars qui détournent les serveurs de messagerie pour envoyer des spams faisant la promotion de sites pornographiques. Si nous utilisons le filtrage pour réduire leurs options à des mails comme celui ci-dessus, cela devrait à peu près mettre les spameurs à l'extrême "légitime" du spectre hors d'activité ; ils se sentent obligés par diverses lois de l'État d'inclure un modèle standard sur la raison pour laquelle leur spam n'est pas du spam, et comment annuler votre "abonnement", et ce genre de message est facile à reconnaître.

(J'avais l'habitude de penser qu'il était naïf de croire que des lois plus strictes réduiraient le spam. Maintenant, je pense que bien que des lois plus strictes ne puissent pas réduire la quantité de spam que les spameurs envoient, elles peuvent certainement aider les filtres à réduire la quantité de spam que les destinataires voient réellement.)

Tout au long du spectre, si vous limitez les arguments de vente que les spameurs peuvent faire, vous aurez inévitablement tendance à les mettre hors d'affaire. Ce mot business est important à retenir. Les spameurs sont des hommes d'affaires. Ils envoient du spam parce que ça marche. Cela fonctionne parce que bien que le taux de réponse soit abominablement bas (au mieux 15 par million, contre 3000 par million pour un envoi de catalogue), le coût, pour eux, n'est pratiquement rien. Le coût est énorme pour les destinataires, environ 5 semaines-homme pour chaque million de destinataires qui passent une seconde à supprimer le spam, mais le spameur n'a pas à payer cela.

L'envoi de spam coûte quelque chose au spammer cependant [2]. Ainsi, plus nous pouvons obtenir le taux de réponse - que ce soit en filtrant ou en utilisant des filtres pour forcer les spameurs à diluer leurs présentations - moins d'entreprises trouveront qu'il vaut la peine d'envoyer du spam.

La raison pour laquelle les spameurs utilisent le genre d'argumentaires de vente qu'ils utilisent est d'augmenter les taux de réponse. C'est peut-être encore plus dégoûtant que d'entrer dans l'esprit d'un spameur, mais jetons un coup d'œil rapide à l'intérieur de l'esprit de quelqu'un qui répond à un spam. Cette personne est soit étonnamment crédule, soit profondément dans le déni de ses intérêts sexuels. Dans les deux cas, aussi répugnant ou idiot que le spam nous semble, c'est excitant pour eux. Les spameurs ne diraient pas ces choses si elles ne semblaient pas excitantes. Et "Checkez ce qui suit" ne va tout simplement pas avoir presque l'attraction avec le destinataire du spam comme le genre de choses que les spameurs disent maintenant. Résultat : s'il ne peut pas contenir des arguments de vente passionnants, le spam devient moins efficace en tant que véhicule de marketing et moins d'entreprises veulent l'utiliser.

C'est la grande victoire à la fin. J'ai commencé à écrire un logiciel de filtrage du spam parce que je ne voulais plus avoir à regarder ce genre de choses. Mais si nous sommes assez bons pour filtrer les spams, ils cesseront de fonctionner, et les spameurs cesseront enfin de les envoyer.

De toutes les approches de la lutte contre le spam, du logiciel aux lois, je crois que le filtrage bayésien sera le plus efficace. Mais je pense aussi que plus nous entreprenons d'efforts antispam différents, mieux c'est, car toute mesure qui limite les spameurs aura tendance à faciliter le filtrage. Et même dans le monde du filtrage basé sur le contenu, je pense que ce serait une bonne chose s'il y a de nombreux types de logiciels différents utilisés simultanément. Plus il y a de filtres différents, plus il sera difficile pour les spameurs d'accorder les spams pour les traverser.

Chapitre 9

Le Goût Pour les Créateurs

“Les objections esthétiques de Copernic à [équants] ont fourni un motif essentiel pour son rejet du système ptolémaïque...”

Thomas Kuhn, *La Révolution Copernicienne*

“Nous avions tous été entraînés par Kelly Johnson et étions fanatiques dans son insistance sur le fait qu'un avion qui avait l'air beau volerait de la même manière.”

Ben Rich, *Skunk Works*

“La beauté est le premier test : il n'y a pas de place permanente dans ce monde pour les mathématiques laides.

G. H. Hardy, *A Mathematician's Apology*

Je parlais récemment à un ami qui enseigne au MIT. Son domaine est en plein essor maintenant et, chaque année, il est inondé de candidatures d'étudiants potentiels aux cycles supérieurs. « Beaucoup d'entre eux semblent intelligents », a-t-il déclaré. « Ce que je ne peux pas dire, c'est s'ils ont un quelconque goût. »

Goût. Vous n'entendez pas beaucoup ce mot maintenant. Et pourtant, nous avons toujours besoin du concept sous-jacent, quel que soit le nom que nous l'appelons. Ce que mon ami voulait dire, c'est qu'il voulait des étudiants qui n'étaient pas seulement de bons techniciens, mais qui pouvaient utiliser leurs connaissances techniques pour concevoir de belles choses.

Les mathématiciens appellent le bon travail "beau", et donc, maintenant ou dans le passé, ont des scientifiques, des ingénieurs, des musiciens, des architectes, des designers, écrivains et peintres. Est-ce juste une coïncidence qu'ils aient utilisé le même mot, ou y a-t-il un chevauchement dans ce qu'ils voulaient dire ? S'il y a un chevauchement, pouvons-nous utiliser les découvertes d'un domaine sur la beauté pour nous aider dans un autre ?

Pour ceux d'entre nous qui conçoivent des choses, ce ne sont pas seulement des questions théoriques. S'il existe une chose telle que la beauté, nous devons être en mesure de la reconnaître. Nous devons avoir bon goût pour faire de bonnes choses. Au lieu de traiter la beauté comme une abstraction aérée, à traiter ou à éviter en fonction de ce que l'on pense des abstractions aérées, essayons de la considérer comme une question pratique : *comment faites-vous de bonnes choses ?*

Si vous mentionnez le goût de nos jours, beaucoup de gens vous diront que « le goût est subjectif ». Ils y croient parce que c'est vraiment ce qu'ils ressentent. Quand ils aiment quelque chose, ils n'ont aucune idée de pourquoi. C'est peut-être parce que c'est beau, ou parce que leur mère en avait un, ou parce qu'ils ont vu une star de cinéma avec un dans un magazine, ou parce qu'ils savent que c'est cher. Leurs pensées sont un enchevêtrement d'impulsions non examinées.

La plupart d'entre nous ont été encouragés, en tant qu'enfants, à laisser cet enchevêtrement non examiné. Si vous vous moquiez de votre petit frère pour avoir coloré les gens en vert dans son livre de coloriage, votre mère était susceptible de vous dire quelque chose comme "vous aimez le faire à votre façon et il aime le faire à sa façon".

À ce stade, votre mère n'essayait pas de vous enseigner des vérités importantes sur l'esthétique. Elle essayait de vous amener tous les deux à arrêter de vous chamailler.

Comme beaucoup de demi-vérités que les adultes nous ont dites, celle-ci contre-dicte d'autres choses qu'ils nous ont dites. Après vous avoir appris que le goût n'est qu'une question de préférence personnelle, ils vous ont emmené au musée et vous ont dit que vous devriez faire preuve d'attention parce que Léonard de Vinci est un grand artiste.

Qu'est-ce qui passe par la tête de l'enfant à ce stade ? Que pense-t-il que signifie « grand artiste » ? Après avoir été dit pendant des années que tout le monde aime juste faire les choses à sa façon, il est peu probable d'aller directement à la conclusion qu'un grand artiste est quelqu'un dont le travail est meilleur que celui des autres. Une théorie beaucoup plus probable, dans son

modèle ptolémaïque de l'univers, est qu'un grand artiste est quelque chose de bon pour vous, comme le brocoli, parce que quelqu'un l'a dit dans un livre.

Dire que le goût n'est qu'une préférence personnelle est un bon moyen de prévenir les litiges. Le problème, c'est que ce n'est pas vrai. Vous le ressentez lorsque vous commencez à concevoir des choses.

Quel que soit le travail que les gens font, ils veulent naturellement faire mieux. Les joueurs de football aiment gagner des matchs. Les PDG aiment augmenter leurs bénéfices. C'est une question de fierté, et un vrai plaisir, de s'améliorer dans votre travail. Mais si votre travail consiste à concevoir des choses, et qu'il n'y a pas de beauté, alors il n'y a *aucun moyen de vous améliorer dans votre travail*. Si le goût n'est qu'une préférence personnelle, alors celui de tout le monde est déjà parfait : vous aimez ce que vous voulez, et c'est tout.

Comme dans tout travail, à mesure que vous continuez à concevoir des choses, vous vous améliorerez. Vos goûts vont changer. Et, comme tous ceux qui s'améliorent dans leur travail, vous saurez que vous vous améliorez. Si c'est le cas, vos anciens goûts n'étaient pas seulement différents, mais pires. Pouf va à l'axiome que le goût ne peut pas être faux.

Le relativisme est à la mode en ce moment, et cela peut vous empêcher de penser au goût, même si le vôtre grandit. Mais si vous sortez du placard et admettez, au moins à vous-même, qu'il existe une bonne conception, alors vous pouvez commencer à l'étudier en détail. Comment votre goût a-t-il changé ? Lorsque vous avez fait des erreurs, qu'est-ce qui vous a poussé à les faire ? Qu'est-ce que d'autres personnes ont appris sur le design ?

Une fois que vous commencez à examiner la question, il est surprenant de voir à quel point les idées de beauté des différents domaines ont en commun. Les mêmes principes de bonne conception se récoltent encore et encore.

Un bon design est simple. Vous entendez cela des mathématiques à la peinture. En mathématiques, cela signifie qu'une preuve plus courte a tendance à être meilleure. En ce qui concerne les axiomes, en particulier, moins égale plus. Cela signifie à peu près la même chose dans la programmation. Pour les architectes et les concepteurs, cela signifie que la beauté devrait dépendre de quelques

éléments structurels soigneusement choisis plutôt que d'une profusion d'aménagements superficiels. (L'ornement n'est pas mauvais en soi, seulement lorsqu'il est camouflé sur une forme insipide.) De même, en peinture, une nature morte de quelques objets soigneusement observés et solidement modelés aura tendance à être plus intéressant qu'un étirement de peinture flashy mais répétitivement répétitive de, par exemple, un collier en dentelle. Par écrit, cela signifie : dites ce que vous voulez dire et dites-le brièvement.

Il semble étrange de devoir mettre l'accent sur la simplicité. On pourrait penser que simple serait la valeur par défaut. Orné, c'est plus de travail. Mais quelque chose semble venir des gens quand ils essaient d'être créatifs. Les écrivains débutants adoptent un ton pompeux qui ne ressemble en rien à la façon dont ils parlent. Les designers essayant d'être un recours artistique aux swooshes et aux boucles. Les peintres découvrent qu'ils sont expressionnistes. C'est de l'évasion. Sous les longs mots ou les coups de pinceau « expressifs », il ne se passe pas grand-chose, et c'est effrayant.

Lorsque vous êtes forcé d'être simple, vous êtes obligé de faire face au vrai problème. Lorsque vous ne pouvez pas livrer d'ornement, vous devez livrer de la substance.

Un bon design est intemporel. En mathématiques, chaque preuve est intemporelle à moins qu'elle ne contienne une erreur. Alors, que veut dire Hardy quand il dit qu'il n'y a pas de place permanente pour les mathématiques laides ? Il veut dire la même chose que Kelly Johnson : si quelque chose est laid, ce ne peut pas être la meilleure solution. Il doit y en avoir un meilleur, et finalement quelqu'un d'autre le découvrira.

Viser l'intemporalité est un moyen de vous faire trouver la meilleure réponse : si vous pouvez imaginer quelqu'un vous surpasser, vous devriez le faire vous-même. Certains des plus grands maîtres l'ont si bien fait qu'ils ont laissé peu de place à ceux qui sont venus après. Chaque graveur depuis Dürer souffre de la comparaison.

Viser l'intemporalité est également un moyen d'échapper à l'emprise de la mode. Les modes changent presque par définition avec le temps, donc si vous

pouvez faire quelque chose qui aura encore l'air bien loin dans le futur, alors son attrait doit dériver plus du mérite que de la mode.

Curieusement, si vous voulez faire quelque chose qui plaira aux générations futures, une façon de le faire est d'essayer de faire appel aux générations passées. Il est difficile de deviner à quoi ressemblera l'avenir, mais nous pouvons être sûrs que ce sera comme le passé en ne prenant soin de rien pour les modes présentes. Donc, si vous pouvez faire quelque chose qui plaira aux gens aujourd'hui et qui aurait également fait appel aux gens en 1500, il y a de fortes chances que cela plaise aux gens en 2500.

Un bon design résout le bon problème. Le poêle typique a quatre brûleurs disposés en carré et un cadran pour contrôler chacun d'eux. Comment arrangez-vous les cadrants ? La réponse la plus simple est de les mettre dans une rangée. Mais c'est une réponse simple à la mauvaise question. Les cadrants sont pour les humains, et si vous les mettez dans une rangée, l'humain malchanceux devra s'arrêter et réfléchir à chaque fois à quel cadran correspond à quel brûleur. Mieux vaut disposer les cadrants dans un carré comme les brûleurs.

Beaucoup de mauvais design est industriel, mais malavisé. Au milieu du XXe siècle, il y avait une mode pour mettre du texte en police sans-serif. Ces polices sont plus proches des formes de lettres pures et sous-jacentes. Mais dans le texte, ce n'est pas le problème que vous essayez de résoudre. Pour la lisibilité, il est plus important que les lettres soient faciles à distinguer. Il peut avoir l'air victorien, mais un g minuscule Times Roman est facile à distinguer à partir d'un y minuscule.

Les problèmes peuvent être améliorés ainsi que des solutions. Dans les logiciels, un problème insoluble peut généralement être remplacé par un problème équivalent facile à résoudre. La physique a progressé plus rapidement au fur et à mesure que le problème prévoyait un comportement observable, au lieu de le réconcilier avec les Écritures.

Un bon design est suggestif. Les romans de Jane Austen ne contiennent presque aucune description ; au lieu de vous dire à quoi tout ressemble, elle raconte son histoire si bien que vous imaginez la scène par vous-même.



Porsche 911E, 1973

De même, une peinture qui suggère est généralement plus engageante qu'une peinture qui le dit. Chacun invente sa propre histoire sur *La Joconde*.

En architecture et en design, ce principe signifie qu'un bâtiment ou un objet doit vous permettre de l'utiliser comme vous le souhaitez : un bon bâtiment, par exemple, servira de toile de fond à la vie que les gens veulent y mener, au lieu de les faire vivre comme s'ils exécutaient un programme écrit par l'architecte.

Dans le logiciel, cela signifie que vous devez donner aux utilisateurs quelques éléments de base qu'ils peuvent combiner comme ils le souhaitent, comme Lego. En mathématiques, cela signifie qu'une preuve qui devient la base de beaucoup de nouveaux travaux est préférable à une preuve qui était difficile, mais qui ne conduit pas à de futures découvertes. En sciences en général, la citation est considérée comme un indicateur approximatif du mérite.

Un bon design est souvent un peu drôle. Celui-ci n'est peut-être pas vrai. Mais les gravures de Dürer et la chaise Womb de Saarinen, le Panthéon et la

Porsche 911 originale me semblent tous un peu drôles. Le théorème d'incomplétude de Gödel semble être une blague pratique.

Je pense que c'est parce que l'humour est lié à la force. Avoir le sens de l'humour, c'est être fort : garder son sens de l'humour, c'est se débarrasser des malheurs, et perdre son sens de l'humour, c'est être blessé par eux. Et donc la marque - ou du moins la prérogative - de la force est de ne pas se prendre trop au sérieux. Les confiants, comme les hirondelles, semblent souvent se moquer légèrement de l'ensemble du processus, comme le fait Hitchcock dans ses films ou Bruegel dans ses peintures (ou Shakespeare, d'ailleurs).

Un bon design n'a peut-être pas besoin d'être drôle, mais il est difficile d'imaginer quelque chose qui pourrait être appelé sans humour étant aussi un bon design.

Un bon design est difficile. Si vous regardez les gens qui ont fait un excellent travail, une chose qu'ils semblent tous avoir en commun, c'est qu'ils ont travaillé très dur. Si vous ne travaillez pas dur, vous gaspillez probablement votre temps.

Les problèmes difficiles appellent de grands efforts. En mathématiques, les épreuves difficiles nécessitent des solutions ingénieuses, et celles-ci ont tendance à être intéressantes. Idem en ingénierie.

Lorsque vous devez escalader une montagne, vous jetez tout ce qui n'est pas nécessaire dans votre sac. Et donc un architecte qui doit construire sur un site difficile, ou un petit budget, trouvera qu'il est obligé de produire un design élégant. Les modes et les fioritures sont mises de côté par la tâche difficile de résoudre le problème.

Tous les types de dur ne sont pas bons. Il y a une bonne douleur et une mauvaise douleur. Vous voulez le genre de douleur que vous ressentez en courant, pas le genre de douleur que vous ressentez en marchant sur un clou. Un problème difficile pourrait être bon pour un concepteur, mais un client inconstant ou des matériaux peu fiables ne le serait pas.

Dans l'art, la plus haute place a traditionnellement été donnée aux peintures de personnes. Il y a quelque chose dans cette tradition, et pas seulement parce que les images de visages appuient sur des boutons dans notre cerveau que d'autres images ne le font pas. Nous sommes si bons à regarder les visages que nous forçons quiconque les attire à travailler dur pour nous satisfaire. Si vous dessinez un arbre et que vous changez l'angle d'une branche de cinq degrés, personne ne le fera savoir. Lorsque vous changez l'angle de l'œil de quelqu'un de cinq degrés, les gens le remarquent.

Lorsque les concepteurs du Bauhaus ont adopté la "forme suit la fonction" de Sullivan, ce qu'ils signifient était, la forme *devrait* suivre la fonction [1]. Et si la fonction est assez difficile, la forme est forcée de la suivre, car il n'y a pas d'effort à ménager pour l'erreur. Les animaux sauvages sont beaux parce qu'ils ont une vie difficile.

Un bon design a l'air facile. Comme les grands athlètes, les grands designers le rendent facile. La plupart du temps, c'est une illusion. Le ton facile et conversationnel d'une bonne écriture n'arrive qu'à la huitième réécriture.

En science et en ingénierie, certaines des plus grandes découvertes semblent si simples que vous vous dites que j'aurais pu y penser. Celui qui l'a découvert a le droit de répondre, pourquoi ne l'avez-vous pas fait ?

Certaines têtes de Léonard de Vinci ne sont que quelques lignes. Vous les regardez et vous pensez que tout ce que vous avez à faire est d'obtenir huit ou dix lignes au bon endroit et vous avez fait ce beau portrait. Eh bien, oui, mais vous devez les mettre *exactement* au bon endroit. La moindre erreur fera s'effondrer le tout.

Les dessins au trait sont en fait le support visuel le plus difficile, car ils exigent presque la perfection. En termes mathématiques, il s'agit d'une solution de forme fermée ; les petits artistes résolvent littéralement les mêmes problèmes par approximation successive. L'une des raisons pour lesquelles les enfants abandonnent de dessiner à l'âge de dix ans environ est qu'ils décident de commencer à dessiner comme des adultes, et l'une des premières choses qu'ils essaient est un dessin au trait d'un visage.

Dans la plupart des domaines, l'apparence de l'aisance semble venir avec la pratique. Peut-être que ce que la pratique fait, c'est d'entraîner votre esprit inconscient à gérer des tâches qui nécessitaient auparavant une pensée consciente. Dans certains cas, vous entraînez littéralement votre corps. Un pianiste expert peut jouer des notes plus rapidement que le cerveau ne peut envoyer des signaux à sa main. De même, un artiste, après un certain temps, peut faire couler la perception visuelle à travers son œil et à travers sa main aussi automatiquement que quelqu'un tapotant son pied à un battement.

Quand les gens parlent d'être dans "la zone", je pense que ce qu'ils veulent dire, c'est que la moelle épinière a la situation sous contrôle. Votre moelle épinière est moins hésitante, et elle libère la pensée consciente pour les problèmes difficiles.

Un bon design utilise la symétrie. La symétrie n'est peut-être qu'un moyen d'atteindre la simplicité, mais elle est suffisamment importante pour être mentionnée seule. La nature l'utilise beaucoup, ce qui est un bon signe.

Il existe deux types de symétrie, la répétition et la récursivité. La récursivité signifie la répétition dans les sous-éléments, comme le motif des veines dans une feuille.

La symétrie est démodée dans certains domaines maintenant, en réaction aux excès du passé. Les architectes ont commencé consciemment à rendre les constructions asymétriques à l'époque victorienne, et dans les années 1920, l'asymétrie était une prémissse explicite de l'architecture moderniste. Même ces bâtiments n'avaient tendance à être asymétriques qu'à propos des axes majeurs, cependant il y avait des centaines de symétries mineures.

En écrivant, vous trouvez une symétrie à tous les niveaux, des phrases à une phrase à l'intrigue d'un roman. Vous trouverez la même chose dans la musique et l'art. Les mosaïques (et certains Cézannes) ont un impact visuel supplémentaire parce que toute l'image est faite des mêmes atomes. La symétrie de composition donne lieu à certaines des peintures les plus mémorables, en particulier lorsque deux moitiés réagissent l'une à l'autre, comme dans la *Création d'Adam* ou *American Gothic*.

En mathématiques et en ingénierie, la récursivité, en particulier, est une grande victoire. Les preuves inductives sont merveilleusement courtes. Dans les logiciels, un problème qui peut être résolu par récursivité est presque toujours mieux résolu de cette façon. La Tour Eiffel semble frappante en partie parce qu'il s'agit d'une solution récursive, une tour sur une tour.

Le danger de la symétrie, et de la répétition en particulier, est qu'il peut être utilisé comme substitut à la pensée.

Un bon design ressemble à la nature. Ce n'est pas tant que la nature ressemblant à la nature est intrinsèquement bonne que la nature a longtemps pour travailler sur le problème. C'est donc un bon signe lorsque votre réponse ressemble à celle de la nature.



Tour Eiffel, 1889. Une tour sur une tour.

Ce n'est pas tricher de copier. Peu de gens nierait qu'une histoire devrait être comme la vie. Travailler à partir de la vie est également un outil précieux en peinture, bien que son rôle ait souvent été mal compris. Le but n'est pas simplement de faire un disque. Le but de la peinture de la vie est qu'elle donne à

votre esprit quelque chose à mâcher : lorsque vos yeux regardent quelque chose, votre main fera un travail plus intéressant.

Imiter la nature marche également dans le domaine de l'ingénierie. Les bateaux ont depuis longtemps des épines et des côtes comme la cage thoracique d'un animal. Dans d'autres cas, nous devrons peut-être attendre une meilleure technologie. Les premiers concepteurs d'avions se sont trompés à concevoir des avions qui ressemblaient à des oiseaux, parce qu'ils n'avaient pas de matériaux ou de sources d'énergie assez légers, ou de systèmes de contrôle assez sophistiqués, pour des machines qui volaient comme des oiseaux [2]. Mais je pouvais imaginer de petits avions de reconnaissance sans pilote volant comme des oiseaux en cinquante ans.



Léonard de Vinci, étude d'un cheval d'élevage, 1481-99.

Maintenant que nous avons assez de puissance informatique, nous pouvons imiter la méthode de la nature ainsi que ses résultats. Les algorithmes génétiques peuvent nous permettre de créer des choses trop complexes pour être conçues dans un sens ordinaire.

Un bon design est une redesign. Il est rare de bien faire les choses la première fois. Les experts s'attendent à jeter un peu de travail précoce. Ils prévoient que les plans changent.

Il faut de la confiance pour jeter le travail. Vous devez être capable de penser, *il y a plus d'où cela vient*. Lorsque les gens commencent à dessiner pour la première fois, par exemple, ils sont souvent réticents à refaire des parties qui ne sont pas correctes. Ils ont l'impression d'avoir eu de la chance d'aller aussi loin, et s'ils essaient de refaire quelque chose, cela s'aggravera. Au lieu de cela, ils se convainquent que le dessin n'est pas si mauvais, vraiment - en fait, peut-être qu'ils voulaient qu'il ait l'air comme ça.

Territoire dangereux. Si quoi que ce soit, vous devriez cultiver l'insatisfaction. Dans les dessins de Léonard de Vinci, il y a souvent cinq ou six tentatives pour obtenir une ligne droite. Le dos distinctif de la Porsche 911 n'est apparu que dans la refonte d'un prototype maladroit. Dans les premiers plans de Wright pour le Guggenheim, la moitié droite était un ziggourat ; il l'a inversée pour obtenir la forme actuelle.

Les erreurs sont naturelles. Au lieu de les traiter comme des catastrophes, rendez-les faciles à reconnaître et à réparer. Léonard de Vinci a plus ou moins inventé l'esquisse, comme un moyen de faire en sorte que le dessin porte un plus grand poids d'exploration. Les logiciels open source ont moins de bugs, car ils admettent la possibilité de bugs.

Il est utile d'avoir un médium qui facilite le changement. Lorsque la peinture à l'huile a remplacé la détrempe au XVe siècle, elle a aidé les peintres à traiter des sujets difficiles comme la figure humaine parce que, contrairement à la détrempe, l'huile peut être mélangée et surpeinte.

Un bon design peut copier. Les attitudes à l'égard de la copie font souvent un voyage aller-retour. Un novice imite sans le savoir ; ensuite, il essaie avec ennui d'être original ; enfin, il décide qu'il est plus important d'être juste qu'original.

L'imitation inconsciente est presque une recette pour un mauvais design. Si vous ne savez pas d'où viennent vos idées, vous imitez probablement un imitateur. Raphaël a tellement envahi le goût du milieu du XIXe siècle que presque tous ceux qui essayaient de dessiner l'imitaient, souvent à plusieurs reprises. C'est cela, plus que le propre travail de Raphaël, qui a dérangé les préraphaélites.

Les ambitieux ne se contentent pas d'imiter. La deuxième phase de la croissance du goût est une tentative consciente d'originalité.

Je pense que les plus grands maîtres vont jusqu'à une sorte d'apathie. Ils veulent juste obtenir la bonne réponse, et si une partie de la bonne réponse a déjà été découverte par quelqu'un d'autre, ce n'est pas une raison de ne pas l'utiliser. Ils sont assez confiants pour prendre à qui que ce soit sans sentir que leur propre vision sera perdue dans le processus.



Lockheed SR-71, 1964.

Un bon design est souvent étrange. Certains des meilleurs travaux ont une qualité étrange : Euler's Formula, Bruegel's Hunters in the Snow, le SR-71, Lisp. Ils ne sont pas seulement beaux, mais étrangement beaux.

Je ne sais pas pourquoi. C'est peut-être juste ma propre stupidité. Un ouvre-boîte doit sembler miraculeux à un chien. Peut-être que si j'étais assez

intelligent, cela semblerait la chose la plus naturelle au monde que $e^{i\pi} = -1$. Après tout, c'est nécessairement vrai.

La plupart des qualités que j'ai mentionnées sont des choses qui peuvent être cultivées, mais je ne pense pas que cela fonctionne pour cultiver l'étrangeté. Le mieux que vous puissiez faire est de ne pas l'écraser s'il commence à apparaître. Einstein n'a pas essayé de rendre la relativité étrange. Il a essayé de la rendre vraie, et la vérité s'est avérée étrange.

Dans une école d'art où j'ai déjà étudié, les élèves voulaient surtout développer un style personnel. Mais si vous essayez simplement de faire de bonnes choses, vous le ferez inévitablement d'une manière distinctive, tout comme chaque personne marche d'une manière distincte. Michel-Ange n'essayait pas de peindre comme Michel-Ange. Il essayait juste de bien peindre; il ne pouvait s'empêcher de peindre comme Michel-Ange.

Le seul style qui vaut la peine d'avoir est celui que vous ne pouvez pas aider. Et c'est particulièrement vrai pour l'étrangeté. Il n'y a pas de raccourci. Le passage du Nord-Ouest que les Maniéristes, les romantiques et deux générations de lycéens américains ont recherché ne semble pas exister. La seule façon d'y arriver est de passer par le bien et d'en sortir par l'autre côté.



Les chasseurs dans la neige, de Bruegel, 1565.

Un bon design se fait en morceaux. Les habitants de la Florence du XVe siècle comprenaient Brunelleschi, Ghiberti, Donatello, Masaccio, Filippo Lippi, Fra Angelico, Verrocchio, Botticelli, Léonard de Vinci et Michel-Ange. Milan à l'époque était aussi grande que Florence. Combien d'artistes milanais du XVe siècle pouvez-vous nommer ?

Quelque chose se passait à Florence au XVe siècle. Et cela ne peut pas avoir été génétique, parce que cela ne se produit pas maintenant. Vous devez supposer que quelle que soit la capacité innée que Léonard de Vinci et Michel-Ange avaient, il y avait des gens nés à Milan avec tout autant. Qu'est-il arrivé au Léonard de Vinci milanais ?

Il y a environ mille fois plus de personnes vivantes aux États-Unis en ce moment qu'à Florence au XVe siècle. Un millier de Léonard de Vinci et un millier de Michel-Ange marchent parmi nous. Si l'ADN régnait, nous devrions être accueillis quotidiennement par des merveilles artistiques.

Nous ne le sommes pas, et la raison en est que pour faire du Léonard de Vinci, vous avez besoin de plus que sa capacité innée. Vous avez également besoin de Florence en 1450.

Rien n'est plus puissant qu'une communauté de personnes talentueuses travaillant sur des problèmes connexes. Les gènes comptent peu par rapport à la comparaison : être un Léonard de Vinci génétique n'était pas suffisant pour compenser d'être né près de Milan au lieu de Florence. Aujourd'hui, nous nous déplaçons davantage, mais un excellent travail provient encore de manière disproportionnée de quelques points chauds : le Bauhaus, le Manhattan Project, *The New Yorker*, Lockheed's Skunk Works, Xerox Parc.

À tout moment, il y a quelques sujets brûlants et quelques groupes qui font un excellent travail sur eux, et il est presque impossible de faire du bon travail vous-même si vous êtes trop loin de l'un de ces points. Vous pouvez pousser ou tirer ces tendances dans une certaine mesure, mais vous ne pouvez pas vous en séparer. (Peut-être que vous le pouvez, mais le Milanais Léonard de Vinci ne le pouvait pas.)

Un bon design est souvent audacieux. À chaque période de l'histoire, les gens ont cru des choses qui étaient tout simplement ridicules, et les ont cru si fortement que vous risquiez l'ostracisme ou même la violence en disant le contraire.

Si notre propre temps était différent, ce serait remarquable. Pour autant que je sache, ce n'est pas le cas.

Ce problème affecte non seulement toutes les époques, mais aussi dans une certaine mesure, tous les domaines. Une grande partie de l'art de la Renaissance était à son époque considérée comme étonnamment laïque : selon Vasari, Botticelli s'est repenti et a abandonné la peinture, et Fra Bartolommeo et Lorenzo di Credi ont brûlé une partie de leurs œuvres. La théorie de la relativité d'Einstein a défendu de nombreux physiciens contemporains, et n'a pas été pleinement acceptée pendant des décennies - en France, pas avant les années 50 [3].

L'erreur expérimentale d'aujourd'hui est la nouvelle théorie de demain. Si vous voulez découvrir de grandes nouvelles choses, alors au lieu de fermer les yeux sur les endroits où la sagesse conventionnelle et la vérité ne se rencontrent pas tout à fait, vous devriez y prêter une attention particulière.

En pratique, je pense qu'il est plus facile de voir la laideur que d'imaginer la beauté. La plupart des gens qui ont fait de belles choses semblent l'avoir fait en réparant quelque chose qu'ils pensaient laid. Un excellent travail semble se produire parce que quelqu'un voit quelque chose et pense que *je pourrais faire mieux que ça*. Giotto a vu des madones traditionnelles byzantines peintes selon une formule qui avait satisfait tout le monde pendant des siècles, et pour lui, elles semblaient en bois et contre nature. Copernic était tellement troublé par un hack que tous ses contemporains pouvaient tolérer qu'il pensait qu'il devait y avoir une meilleure solution.

L'intolérance à la laideur n'est pas suffisante en soi. Vous devez bien comprendre un domaine avant de développer un bon nez pour ce qui doit être réparé. Vous devez faire vos devoirs. Mais au fur et à mesure que vous devenez un expert dans un domaine, vous commencerez à entendre de petites voix dire : *Quel hack ! Il doit y avoir un meilleur moyen*. N'ignorez pas ces voix.

Cultivez-les. La recette d'un excellent travail est : un goût très exigeant, plus la capacité de le satisfaire

Chapitre 10

Les Langages de Programmation Expliqués

Toute machine a une liste de choses que vous pouvez lui dire de faire. Parfois, la liste est courte. Il n'y a que deux choses que je peux faire à ma bouilloire électronique : l'allumer et l'éteindre. Mon lecteur de CD est plus compliqué. En plus de l'allumer et de l'éteindre, je peux monter et descendre le volume, lui dire de jouer ou de faire une pause, reculer ou avancer une chanson, et lui demander de jouer des chansons dans un ordre aléatoire.

Comme tout autre type de machine, un ordinateur a une liste de choses qu'il peut faire. Par exemple, on peut dire à chaque ordinateur d'ajouter deux nombres. La liste complète des choses qu'un ordinateur peut faire est son langage machine.

Le Langage Machine

Lorsque les ordinateurs ont été inventés pour la première fois, tous les programmes devaient être écrits sous forme de séquences d'instructions en langage machine. Peu de temps après, ils ont commencé à être écrits sous une forme un peu plus pratique appelée *langage assembleur*. En langage assembleur, la liste des commandes est la même, mais vous pouvez utiliser des noms plus adaptés aux programmeurs. Au lieu de faire référence à l'instruction d'ajout comme *11001101*, qui est ce que la machine pourrait l'appeler, vous pouvez dire *add*.

Le problème avec le langage machine/assemblage est que la plupart des ordinateurs ne peuvent faire que des choses très simples. Par exemple, supposons que vous vouliez dire à un ordinateur de bip 10 fois. Il est peu probable qu'il y ait une instruction de machine pour faire quelque chose *n* fois. Donc, si vous vouliez dire à un ordinateur de faire quelque chose 10 fois en utilisant les instructions réelles de la machine, vous devez dire quelque chose d'équivalent à :

```
put the number 10 in memory location 0
a if location 0 is negative, go to line b
beep
subtract 1 from the number in location 0
go to line a
b ...rest of program...
```

Si vous devez faire autant de travail pour faire sonner la machine 10 fois, imaginez le travail d'écrire quelque chose comme un processus de mots ou une feuille de calcul.

Et au fait, jetez un autre coup d'œil au programme. Y aura-t-il un bip dix fois ? Non, onze. Dans la première ligne, j'aurais dû dire 9 au lieu de 10. J'ai délibérément mis un bug dans notre exemple pour illustrer un point important sur les langues. Plus vous avez à dire pour faire quelque chose, plus il est difficile de voir des bugs.

Langues de haut niveau

Imaginez que vous deviez produire des programmes en langage assembleur, mais que vous aviez un assistant pour faire tout le sale boulot pour vous. Donc, vous pourriez juste écrire quelque chose comme :

dotimes 10 bip

Et votre assistant écrirait le langage d'assemblage pour vous (mais sans bugs).

En fait, c'est ainsi que la plupart des programmeurs travaillent. Sauf que l'assistant n'est pas une personne, mais un *compilateur*. Un compilateur est un programme qui traduit des programmes écrits sous une forme pratique, comme le oneliner ci-dessus, dans le langage simple que le matériel comprend.

Le langage le plus pratique que vous donnez au compilateur est appelé un *langage de haut niveau*. Il vous permet de construire vos programmes à partir de commandes puissantes, comme "faire quelque chose *n* fois" au lieu des minables commandes comme "ajouter deux nombres".

Lorsque vous construisez vos programmes à partir de concepts plus importants, vous n'avez pas besoin d'en utiliser autant. Écrit dans notre langage imaginaire de haut niveau, notre programme n'est qu'un cinquième aussi long. Et s'il y avait une erreur, ce serait facile à voir.

Un autre avantage des langages de haut niveau est qu'ils rendent vos programmes plus *portables*. Différents ordinateurs ont tous des langages machine légèrement différents. Vous ne pouvez pas, en règle générale, prendre un programme en langage machine écrit pour un ordinateur et l'exécuter sur un autre. Si vous écriviez vos programmes en langage machine, vous devriez tous les réécrire pour les exécuter sur un nouvel ordinateur. Si vous utilisez un langage de haut niveau, tout ce que vous avez à réécrire est le compilateur.

Les compilateurs ne sont pas le seul moyen d'implémenter des langages de haut niveau. Vous pouvez également utiliser un interpréteur, qui examine votre programme une pièce à la fois et exécute les commandes de langue chinoise correspondantes, au lieu de traduire le tout en langage machine et de l'exécuter.

Open Source

Le langage de haut niveau que vous donnez au compilateur est également connu sous le nom de code source, et la traduction en langage automatique qu'il génère est appelée code objet. Lorsque vous achetez un logiciel commercial, vous n'obtenez généralement que le code objet. (L'objet du code est si difficile à lire qu'il est efficacement crypté, protégeant ainsi les secrets commerciaux de l'entreprise.) Mais ces derniers temps, il existe une approche alternative : les logiciels *open source*, où vous obtenez également le code source, et vous êtes libre de le modifier si vous le souhaitez.

Il y a une réelle différence entre les deux modèles. L'*open source* vous donne beaucoup plus de contrôle. Lorsque vous utilisez des logiciels *open source* et que vous voulez comprendre ce qu'ils font, vous pouvez lire le code source et le découvrir. Si vous le souhaitez, vous pouvez même modifier le logiciel et le recompiler.

L'une des raisons pour lesquelles vous voudrez peut-être le faire est de corriger un bug. Vous ne pouvez pas corriger les bugs dans Microsoft Windows, par exemple, parce que vous n'avez pas le code source. (En théorie, vous pourrez pirater le code objet, mais en pratique, c'est très difficile. C'est aussi probablement interdit par le contrat de licence.) Cela peut être un vrai problème. Lorsqu'un nouveau trou de sécurité est découvert dans Windows, vous devez attendre que Microsoft publie un correctif. Et les failles de sécurité sont au moins réparées rapidement. Si le bug ne fait que paralyser votre ordinateur de temps en temps, vous devrez peut-être attendre la prochaine version complète pour qu'il soit corrigé.

Mais l'avantage de l'open source n'est pas seulement que vous pouvez le réparer quand vous en avez besoin. C'est que tout le monde peut. Les logiciels open source sont comme un document qui a fait l'objet d'un examen par les pairs. Beaucoup de personnes intelligentes ont examiné le code source des systèmes d'exploitation open source comme Linux et FreeBSD et ont déjà trouvé la plupart des bugs. Alors que Windows est aussi fiable que l'assurance qualité des grandes entreprises peut le faire.

Les défenseurs de l'open source sont parfois considérés comme des fous qui sont contre l'idée de propriété en général. Quelques-uns le sont. Mais je ne suis certainement pas contre l'idée de propriété, et pourtant je serais très réticent à installer un logiciel pour lequel je n'avais pas le code source. L'utilisateur final moyen n'a peut-être pas besoin du code source de son traitement de texte, mais lorsque vous avez vraiment besoin de fiabilité, il y a de solides raisons d'ingénierie pour insister sur l'open source.

Guerres linguistiques

La plupart des programmeurs, la plupart du temps, programmait dans des langages de haut niveau. Peu utilisent le langage assembleur maintenant. Le temps d'ordinateur est venu beaucoup moins cher, tandis que le temps de programmeur est plus cher que jamais, de sorte qu'il vaut rarement la peine d'écrire des programmes dans un langage aussi simple. Vous pourriez le faire dans quelques parties critiques, par exemple, d'un jeu d'ordinateur, où vous vouliez microgérer le matériel pour extraire ce dernier incrément de vitesse.

Fortran, Lisp, Cobol, Basic, C, Pascal, Smalltalk, C++, Java, Perl et Python sont tous des langages de haut niveau. Ce ne sont que quelques-uns des plus connus. Il existe littéralement des centaines de langages différents de haut niveau. Et contrairement aux langages machine, qui offrent tous des ensembles d'instructions similaires, ces langages de haut niveau vous donnent des concepts assez différents pour construire des programmes.

Alors, lequel utilisez-vous ? Ah, eh bien, il y a beaucoup de désaccord à ce sujet. Une partie du problème est que si vous utilisez un langage assez longtemps, vous commencez à y penser. Ainsi, tout langage qui est substantiellement différent semble terriblement gênant, même s'il n'y a rien d'intrinsèquement mauvais. Les jugements des programmeurs inexpérimentés sur les mérites relatifs des langages de programmation sont souvent faussés par cet effet.

D'autres hackers, peut-être désireux de paraître sophistiqués, vous diront que tous les langages sont fondamentalement les mêmes. J'ai programmé toutes sortes de langages, a dit le vieux hacker coriace alors qu'il s'approchait du bar, et peu importe ce que vous utilisez. Ce qui compte, c'est de savoir si vous avez les bonnes choses. Ou quelque chose dans ce sens.

C'est absurde, bien sûr. Il y a un monde de différence entre, par exemple, Fortran I et la dernière version de Perl - ou d'ailleurs entre les premières versions de Perl et la dernière version de Perl. Mais le vieux hacker coriace peut lui-même croire ce qu'il dit. Il est possible d'écrire les mêmes programmes primitifs de type Pascal dans presque tous les langages. Si jamais vous ne mangez que chez McDonald's, il semblera que la nourriture soit à peu près la même dans tous les pays.

Certains hackers préfèrent le langage auquel ils sont habitués et n'aiment rien d'autre. D'autres disent que tous les langages sont les mêmes. La vérité se situe quelque part entre ces deux extrêmes. Les langages diffèrent, mais il est difficile de dire avec certitude lesquels sont les meilleurs. Le domaine est toujours en évolution.

Abstraction

Tout comme les langages de haut niveau sont plus abstraits que les langages d'assemblage, certains langages de haut niveau sont plus abstraits que d'autres. Par exemple, C est un niveau assez bas, presque un langage d'assemblage portable, tandis que Lisp est de très haut niveau.

Si les langages de haut niveau sont meilleurs pour la programmation que la langue d'assemblage, alors vous pouvez vous attendre à ce que plus le langage est de haut niveau, mieux c'est. Normalement, oui, mais pas toujours. Un langage peut être très abstrait, mais offrir de mauvaises abstractions. Je pense que cela se produit dans Prolog, par exemple. Il a des abstractions fabuleusement puissantes pour résoudre environ 2 % des problèmes, et le reste du temps, vous vous penchez en arrière pour abuser de ces abstractions pour écrire des programmes Pascal de facto.

Une autre raison pour laquelle vous voudrez peut-être utiliser un langage de niveau inférieur est l'efficacité. Si vous avez besoin que le code soit super rapide, il est préférable de rester près de la machine. La plupart des systèmes d'exploitation sont écrits en C, et ce n'est pas une coïncidence. Au fur et à mesure que le matériel devient plus rapide, il y a moins de pression pour écrire des applications dans des langages aussi bas que C, mais tout le monde semble toujours vouloir que les systèmes d'exploitation soient aussi rapides que possible. (Ou peut-être veulent-ils que la perspective d'attaques de débordement de mémoire les maintienne en alerte [1].)

Ceintures de sécurité ou Menottes ?

Le plus grand débat en conception de langage est probablement celui entre ceux qui pensent qu'un langage devrait empêcher les programmeurs de faire des choses stupides, et ceux qui pensent que les programmeurs devraient être autorisés à faire ce qu'ils veulent. Java est dans l'ancien camp, et Perl dans le second. (Il n'est pas surprenant que le DoD soit important sur Java.)

Les partisans des langages permissifs ridiculisent l'autre type de langages "B&D" (bondage et discipline), avec l'implication plutôt imprudente que ceux qui aiment programmer sont inférieurs. Je ne sais pas ce que l'autre côté appelle

des langages comme Perl. Peut-être ne sont-ils pas du genre à inventer des noms amusants pour l'opposition.

Le débat se résout en plusieurs plus petits, car il y a plusieurs façons d'empêcher les programmeurs de faire des choses stupides. L'une des questions les plus actives en ce moment est la dactylographie statique par rapport à la dactylographie dynamique. Dans un langage orthographié statiquement, vous devez connaître le type de valeurs que chaque variable peut avoir au moment où vous écrivez le programme. Avec la saisie dynamique, vous pouvez définir n'importe quelle variable à n'importe quelle valeur, quand vous le souhaitez.

Les partisans du typage statique soutiennent qu'il aide à prévenir les bugs et aide les compilateurs à générer du code rapide (tous deux vrais). Les partisans de la dactylographie dynamique soutiennent que la dactylographie statique limite les programmes que vous pouvez écrire (également vrai). Je préfère la dactylographie dynamique. Je déteste un langage qui me dit quoi faire. Mais certaines personnes intelligentes semblent aimer l'écriture statique, donc la question reste ouverte.

OO

Un autre grand sujet en ce moment est la programmation *orientée objet*. Cela signifie une façon différente d'organiser les programmes. Supposons que vous vouliez écrire un programme pour trouver les zones des figures bidimensionnelles. Au début, il n'a qu'à connaître les cercles et les carrés. Une façon de le faire serait d'écrire un seul morceau de code, dans lequel vous testez si on vous pose des questions sur un cercle ou un carré, puis d'utiliser la formule correspondante pour trouver la zone. La façon orientée objet d'écrire ce programme serait de créer deux *classes*, un cercle et un carré, puis d'attacher à chaque classe un extrait de code (appelé *méthode*) pour trouver la zone de ce type de figure. Lorsque vous avez besoin de trouver la zone de quelque chose, vous demandez quelle est sa classe, récupérez la méthode correspondante et exécutez-ci pour obtenir la réponse.

Ces deux cas peuvent sembler très similaires, et en effet, ce qui se passe lorsque vous exécutez le code est à peu près le même. (Pas étonnamment, puisque vous résolvez le même problème.) Mais le code peut finir par être tout à

fait différent. Dans la version orientée objet, le code pour trouver les zones des carrés et des cercles peut même se retrouver dans différents fichiers, une partie dans le fichier contenant toutes les choses à faire avec les cercles, et l'autre dans le fichier contenant les choses à faire avec les carrés.

L'avantage de l'approche orientée objet est que si vous voulez changer le programme pour trouver la zone, par exemple, des triangles, vous ajoutez simplement un autre morceau de code pour eux, et vous n'avez même pas à regarder le reste. L'inconvénient, selon les critiques, c'est que l'ajout de choses sans regarder ce qui était déjà là a tendance à produire les mêmes résultats dans les programmes que dans les bâtiments.

Le débat sur la programmation orientée objet n'est pas aussi clair que celui sur la dactylographie statique par rapport à la dactylographie dynamique. En tapant, vous devez choisir l'un ou l'autre. Mais l'orientation objet d'un langage est une question de degré. En effet, il y a deux sens de l'orientation objet : certains langages sont orientés objet dans le sens qu'ils vous laissent programmer dans ce style, et d'autres dans le sens où ils vous obligent à le faire.

Je vois peu d'avantages dans ce dernier. Sûrement un langage qui vous permet de faire x est au moins aussi bon qu'un langage qui vous y oblige. Donc, en ce qui concerne les *langues*, au moins, nous pouvons peaufiner cette question. Bien sûr, utilisez un langage qui vous permet d'écrire des programmes orientés objet. Que vous le vouliez réellement, cela devient une question distincte.

Renaissance

Une chose sur laquelle je pense que tout le monde dans le secteur des langages sera d'accord, c'est qu'il y a beaucoup de nouveaux langages de programmation ces derniers temps. Jusqu'aux années 1980, seules les institutions pouvaient se permettre le matériel nécessaire pour développer des langages de programmation, et la plupart ont donc été conçus par des professeurs ou des chercheurs de grandes entreprises. Maintenant, un lycéen peut se permettre tout le matériel nécessaire.

Inspiré en grande partie par l'exemple de Larry Wall, le concepteur de Perl, beaucoup de hackers se demandent : pourquoi ne puis-je pas concevoir

mon propre langage ? Ceux qui parviennent à exploiter la puissance de la communauté open source peuvent obtenir beaucoup de code écrit pour eux très rapidement.

Le résultat est une sorte de langage que vous pourriez appeler *très lourd* : un langage dont le noyau interne n'est pas très bien conçu, mais qui possède des bibliothèques de code extrêmement puissantes pour résoudre des problèmes spécifiques. (Imaginez un Yugo avec un moteur à réaction boulonné au toit.) Pour les petits problèmes quotidiens que les programmeurs passent tant de temps à résoudre, les bibliothèques sont probablement plus importantes que le langage de base. Et donc ces hybrides étranges sont très utiles, et deviennent en conséquence populaires. Un Yugo avec un moteur à réaction boulonné au toit pourrait en fait fonctionner, tant que vous n'avez pas essayé de prendre un virage dedans [2].

Un autre résultat est une grande variété. Il y a toujours eu beaucoup de variété dans les langages de programmation. Fortran, Lisp et APL diffèrent autant les uns des autres que les étoiles de mer, les ours et les mouches-dragons, et tous ont été conçus avant 1970. Mais les nouveaux langages open source ont certainement poursuivi cette tradition.

Il semble que j'entends parler d'un nouveau langage tous les deux jours. Jonathan Erickson l'a appelé "le langage de programmation renaissance". Une autre expression que les gens utilisent parfois est « les guerres du langage ». Mais il n'y a pas de contradiction ici. La Renaissance était pleine de guerres.

En effet, de nombreux historiens pensent que les guerres ont été sous-produit des forces qui ont créé la Renaissance [3]. La clé de la vigueur de l'Europe a peut-être été le fait qu'elle a été divisée en un certain nombre de petits États concurrents. Ceux-ci étaient assez proches pour que les idées puissent voyager de l'une à l'autre, mais assez indépendants pour qu'aucun dirigeant ne puisse mettre un couvercle sur l'innovation - comme la cour chinoise l'a fait de manière désastreuse lorsqu'elle a interdit le développement de grands navires océaniques.

Il est donc probablement bon que les programmeurs vivent dans un monde post-Babel. Si nous utilisions tous le même langage, ce serait probablement le mauvais.

Chapitre 11

Le Langage des Cent-Ans

Il est difficile de prédire à quoi ressemblera la vie dans cent ans. Il n'y a que quelques choses que nous pouvons dire avec certitude. Nous savons que tout le monde conduira des voitures volantes, que les lois de zonage seront assouplies pour permettre aux bâtiments de plusieurs centaines d'étages, qu'il fera nuit la plupart du temps et que les femmes seront toutes formées aux arts martiaux. Ici, je veux zoomer sur un détail de cette image. Quel type de langage de programmation utiliseront-ils pour écrire le logiciel contrôlant ces voitures volantes ?

Cela vaut la peine d'y penser, non pas tant que nous utiliserons effectivement ces langages que parce que, si nous avons de la chance, nous utilisons des langages sur le chemin qui mène d'un point à un autre.

Je pense que, comme les espèces, les langues formeront des arbres évolutifs, avec des impasses qui se ramifient partout. Nous pouvons déjà voir cela se produire. Cobol, malgré sa popularité, ne semble pas avoir de descendance intellectuelles. C'est une impasse évolutive, une langue néandertale.

Je prédis un futur similaire pour Java. Les gens m'envoient parfois des courriels en disant : "Comment pouvez-vous dire que Java ne s'est pas révélé être un langage réussi ? C'est déjà un langage à succès. » Et j'admetts que c'est le cas, si vous mesurez le succès par l'espace d'étagère occupé par les livres dessus, ou par le nombre d'étudiants de premier cycle qui croient qu'ils doivent l'apprendre pour obtenir un emploi. Quand je dis que Java ne s'est pas révélé être un langage réussi, je veux dire quelque chose de plus spécifique : que Java s'est révélé être une impasse évolutive, comme Cobol.

Ce n'est qu'une supposition. Je me trompe peut-être. Mon but ici n'est pas de casser Java, mais de soulever la question des arbres évolutifs et d'amener les gens à se demander, où sur l'arbre se trouve le langage x ? La raison de poser cette question n'est pas seulement pour que dans cent ans, nos fantômes puissent dire, je vous l'ai dit. C'est parce que rester près des branches principales est une

heuristique utile pour trouver des langages qui seront bons à programmer maintenant.

À tout moment, vous serez probablement le plus heureux sur les branches principales d'un arbre évolutif. Même quand il y avait encore beaucoup de Néandertaliens, il a dû être nul d'en être un. Les Cro-Magnons auraient constamment été en train de venir se battre avec vous et de voler votre nourriture.

La raison pour laquelle je veux savoir à quoi ressembleront les langages dans cent ans, c'est pour savoir sur quelle branche de l'arbre parler maintenant.

L'évolution des langages diffère de l'évolution des espèces parce que les branches peuvent converger. La branche de Fortran, par exemple, semble fusionner avec les descendants d'Algol. En théorie, c'est possible pour les espèces aussi, mais c'est si peu probable que cela ne se soit probablement jamais produit.

La convergence est plus probable pour les langages, en partie parce que l'espace des possibilités est plus petit, et en partie parce que les mutations ne sont pas aléatoires. Les concepteurs de langages intègrent délibérément des idées d'autres langages.

Il est particulièrement utile pour les concepteurs de langages de réfléchir à l'évolution des langages de programmation qui est susceptible de mener, car ils peuvent se diriger en conséquence. Dans ce cas, « rester sur une branche principale » devient plus qu'un moyen de choisir un bon langage. Cela devient une heuristique pour prendre les bonnes décisions en matière de conception du langage.

N'importe quel langage de programmation peut être divisé en deux parties : un ensemble d'opérateurs fondamentaux qui jouent le rôle d'axiomes, et le reste du langage, qui pourrait en principe être écrit en termes de ces opérateurs fondamentaux.

Je pense que les opérateurs fondamentaux sont les facteurs les plus importants dans la survie à long terme d'un langage. Le reste, vous pouvez changer. C'est comme la règle selon laquelle lors de l'achat d'une maison, vous

devez d'abord tenir compte de l'emplacement. Tout le reste, vous pouvez le réparer plus tard, mais vous ne pouvez pas réparer l'emplacement.

Il est important non seulement que les axiomes soient bien choisis, mais aussi qu'il y en ait peu. Les mathématiciens ont toujours ressenti cela à propos des axiomes - moins il y en a, mieux c'est - et je pense qu'ils ont mis le doigt sur quelque chose.

À tout le moins, il doit être utile d'examiner de près le noyau d'un langage pour voir s'il y a des axiomes qui pourraient être éliminés. J'ai trouvé dans ma longue carrière de bidouilleur que le croûtage élève le croûtage, et j'ai vu cela se produire dans les logiciels ainsi que sous les lits et dans les coins des chambres.

J'ai l'impression que les branches principales de l'arbre évolutif passent par les langages qui ont les noyaux les plus petits et les plus propres. Plus vous pouvez écrire d'un langage en soi, mieux c'est.

Bien sûr, je fais une grande supposition en demandant même à quoi ressembleront les langages de programmation dans cent ans. Écrirons-nous même des programmes dans cent ans ? Ne dirons-nous pas simplement aux ordinateurs ce que nous voulons qu'ils fassent ?

Il n'y a pas eu beaucoup de progrès dans ce département jusqu'à présent. Je suppose que dans cent ans, les gens diront toujours aux ordinateurs quoi faire en utilisant des programmes que nous reconnaîtrions comme tels. Il peut y avoir des tâches que nous résolvons maintenant en écrivant des programmes et que dans cent ans, vous n'aurez pas à écrire des programmes à résoudre, mais je pense qu'il y aura encore beaucoup de programmation du type que nous faisons aujourd'hui.

Il peut sembler présomptueux de penser que n'importe qui peut prédire à quoi ressemblera n'importe quelle technologie dans cent ans. Mais rappelons-nous que nous avons déjà près de cinquante ans d'histoire derrière nous. Regarder vers l'avenir cent ans est une idée saisissable lorsque nous considérons à quel point les langages ont évolué lentement au cours des cinquante dernières années.

Les langages évoluent lentement parce qu'elles ne sont pas vraiment des technologies. Les langages sont la notation. Un programme est une description formelle du problème que vous voulez qu'un ordinateur résolve pour vous. Ainsi, le taux d'évolution dans les langages de programmation ressemble plus au taux d'évolution en notation mathématique que, par exemple, au transport ou aux communications. La notation mathématique évolue, mais pas avec les sauts géants que vous voyez dans la technologie.

Quels que soient les ordinateurs dans cent ans, il semble sûr de prédire qu'ils seront beaucoup plus rapides. Si la loi de Moore continue de s'édifier, elle sera 74 quintillions (73 786 976 294 838 206 464) fois plus rapide. C'est un peu difficile à imaginer. Et en effet, la prédiction la plus probable dans le département de la vitesse est peut-être que la loi de Moore cessera de fonctionner. Tout ce qui est censé doubler tous les dix-huit mois semble susceptible de se heurter à une sorte de limite fondamentale. Mais je n'ai aucun mal à croire que les ordinateurs seront beaucoup plus rapides. Même s'ils finissent par être un million de fois plus rapides, cela devrait changer considérablement les règles de base pour les langages de programmation. Entre autres choses, il y aura plus de place pour ce qui serait maintenant considéré comme des langages lents, c'est-à-dire des langages qui ne donnent pas de code très efficace.

Et pourtant, certaines applications exigeront toujours de la vitesse. Certains des problèmes que nous voulons résoudre avec les ordinateurs sont créés par des ordinateurs ; par exemple, la vitesse à laquelle vous devez traiter les images vidéo dépend de la vitesse à laquelle un autre ordinateur peut les générer. Et il y a une autre classe de problèmes qui ont intrinsèquement une capacité illimitée pour absorber les cycles : le rendu d'images, la cryptographie, les simulations.

Si certaines applications peuvent être de plus en plus inefficaces tandis que d'autres continuent d'exiger toute la vitesse que le matériel peut offrir, des ordinateurs plus rapides signifieront que les langages devront couvrir un éventail toujours plus large d'efficacités. Nous avons déjà vu cela se produire. Les implémentations actuelles de certains nouveaux langages populaires sont étonnamment inutiles selon les normes des décennies précédentes.

Ce n'est pas seulement quelque chose qui se passe avec les langages de programmation. C'est une tendance historique générale. Au fur et à mesure que les technologies s'améliorent, chaque génération peut faire des choses que la génération précédente aurait considérées comme du gaspillage. Il y a trente ans, les gens étaient étonnés de voir à quel point nous faisions des appels téléphoniques interurbains. Il y a cent ans, les gens seraient encore plus étonnés qu'un forfait voyage d'un jour de Boston à New York via Memphis existe.

Je peux déjà vous dire ce qui va arriver à tous ces cycles supplémentaires que le matériel plus rapide va nous donner au cours des cent prochaines années. Ils vont presque tous être gaspillés.

J'ai appris à programmer quand la puissance de l'ordinateur était rare. Je me souviens avoir pris tous les espaces de mes programmes de base pour qu'ils s'intègrent dans la mémoire d'un TRS-80 4K. L'idée de tous ces logiciels incroyablement inefficaces qui brûlent des cycles en faisant la même chose encore et encore me semble un peu grossière. Mais je pense que mes intuitions ici sont fausses. Je suis comme quelqu'un qui a grandi pauvre et qui ne supporte pas de dépenser de l'argent même pour quelque chose d'important, comme aller chez le médecin.

Certains types de déchets sont vraiment dégoûtants. Les SUVs, par exemple, seraient sans doute grossiers même s'ils étaient alimentés par un carburant qui ne s'épuiserait jamais et ne générerait aucune pollution. Les SUVs sont grossiers parce qu'ils sont la solution à un problème grossier. (Comment rendre les mini-fourgonnettes plus masculines.) Mais tous les déchets ne sont pas mauvais. Maintenant que nous avons l'infrastructure pour le prendre en charge, compter les minutes de vos appels longue distance commence à sembler tatillon. Si vous avez les ressources, il est plus élégant de considérer tous les appels téléphoniques comme un seul genre de chose, peu importe où se trouve l'autre personne.

Il y a de bons déchets et de mauvais déchets. Je m'intéresse aux bons déchets - ceux où, en dépensant plus, nous pouvons obtenir des conceptions plus simples. Comment allons-nous profiter des opportunités de gaspillage des cycles que nous obtiendrons du nouveau matériel plus rapide ?

Le désir de vitesse est si profondément enraciné en nous, avec nos ordinateurs chétifs, qu'il faudra un effort conscient pour le surmonter. Dans la conception du langage, nous devrions rechercher consciemment des situations où nous pouvons échanger l'efficacité même pour la plus petite augmentation de commodité.

La plupart des structures de données existent en raison de la vitesse. Par exemple, de nombreuses langues d'aujourd'hui ont à la fois des chaînes et des listes. Sur le plan sémantique, les chaînes sont plus ou moins un sous-ensemble de listes dans lesquelles les éléments sont des caractères. Alors, pourquoi avez-vous besoin d'un type de données distinct ? Vous ne le faites pas, vraiment. Les cordes n'existent que pour l'efficacité. Mais c'est boiteux d'encombrer les caractéristiques d'un langage avec des hacks pour que les programmes s'exécutent plus rapidement. Avoir des cordes dans un langage semble être un cas d'opération prématuée.

Si nous pensons au cœur d'un langage comme à un ensemble d'axiomes, il est sûrement grossier d'avoir des axiomes supplémentaires qui n'ajoutent aucun pouvoir expressif, simplement par souci d'efficacité. L'efficacité est importante, mais je ne pense pas que ce soit la bonne façon de l'obtenir.

La bonne façon de résoudre ce problème est de séparer la signification d'un programme des détails de la mise en œuvre. Au lieu d'avoir à la fois des listes et des chaînes, n'ayez que des listes, avec un moyen de donner au compilateur des conseils d'optimisation qui lui permettront de présenter des chaînes en octets contigus si nécessaire [1].

Étant donné que la vitesse n'a pas d'importance dans la majeure partie des programmes, vous n'aurez normalement pas besoin de vous préoccuper de ce genre de micro-gestion. Cela sera de plus en plus vrai au fur et à mesure que les ordinateurs deviendront plus rapides..

Dire moins sur la mise en œuvre devrait également rendre les programmes plus flexibles. Les spécifications changent pendant qu'un programme est écrit, et c'est non seulement inévitable, mais aussi souhaitable.

Le mot "essay" vient du verbe français "essayer". Un essai, au sens original, est quelque chose que vous écrivez pour essayer de comprendre quelque chose. Cela se produit aussi dans les logiciels. Je pense que certains des meilleurs programmes étaient des essais, en ce sens que les auteurs ne savaient pas quand ils ont commencé exactement ce qu'ils essayaient d'écrire.

Les hackers Lisp connaissent déjà l'intérêt d'être flexible avec les structures de données. Nous avons tendance à écrire la première version d'un programme pour qu'il fasse tout avec des listes. Ces versions initiales peuvent être si étonnamment inefficaces qu'il faut un effort conscient pour ne pas penser à ce qu'elles font, tout comme, pour moi au moins, manger un steak nécessite un effort conscient pour ne pas penser d'où il vient.

Ce que les programmeurs dans cent ans chercheront, surtout, c'est un langage où vous pouvez assembler une version 1 incroyablement inefficace d'un programme avec le moins d'effort possible. Au moins, c'est ainsi que nous le décririons en termes actuels. Ce qu'ils diront, c'est qu'ils veulent un langage facile à programmer.

Les logiciels inefficaces ne sont pas grossiers. Ce qui est grossier, c'est un langage qui oblige les programmeurs à faire un travail inutile. La gaspillage du temps du programmeur est la véritable inefficacité, et non le gaspillage du temps de la machine. Cela sera de plus en plus clair au fur et à mesure que les ordinateurs deviendront plus rapides.

Je pense que se débarrasser des ficelles est déjà quelque chose à laquelle nous pourrions supporter de penser. Nous l'avons fait dans Arc, et cela semble être une victoire ; certaines opérations qui seraient difficiles à décrire comme expressions régulières peuvent être facilement décrites comme des fonctions récursives.

Jusqu'où ira cet aplatissement des structures de données ? Je peux penser à des possibilités qui me choquent même, avec mon esprit consciencieusement large. Allons-nous nous débarrasser des tableaux, par exemple ? Après tout, ils ne sont qu'un sous-ensemble de tables de hachage où les clés sont des vecteurs entiers. Allons-nous remplacer les tables de hachage elles-mêmes par des listes ?

Il y a même plus de perspectives choquantes que cela. Logiquement, vous n'avez pas besoin d'avoir une notion distincte des nombres, car vous pouvez les représenter sous forme de listes : l'entier n pourrait être représenté sous la forme d'une liste de n éléments. Vous pouvez faire des maths de cette façon. C'est tout simplement insupportablement inefficace.

Un langage de programmation pourrait-il aller jusqu'à se débarrasser des nombres en tant que type de données fondamental ? Je pose cette question moins comme une question sérieuse que comme une façon de jouer à la poule mouillée avec l'avenir. C'est comme le cas hypothétique d'une force irrésistible rencontrant un objet immobile - ici, une rencontre de mise en œuvre inimaginablement inefficace rencontrant des ressources inimaginablement grandes. Je ne vois pas pourquoi pas. L'avenir est assez long. S'il y a quelque chose que nous pouvons faire pour diminuer le nombre d'axiomes dans le langage de base, cela semblerait être le côté sur lequel parier à l'approche de l'infini. Si l'idée semble encore insupportable dans cent ans, peut-être qu'elle ne le sera pas dans mille ans.

Juste pour être clair à ce sujet, je ne propose pas que tous les calculs numériques soient réellement effectués à l'aide de listes. Je propose que le langage de base, avant toute notation supplémentaire sur la mise en œuvre, soit défini de cette façon. En pratique, tout programme qui voudrait faire une quantité quelconque de mathématiques représenterait probablement des nombres en binaire, mais ce serait une optimisation, et ne ferait pas partie de la sémantique du langage de base.

Entre l'application et le matériel. C'est aussi une tendance que nous voyons déjà se produire : de nombreux langages récents sont compilés en code d'octets. Bill Woods m'a dit un jour que, en règle générale, chaque couche d'interprétation coûte un facteur dix en vitesse. Ce coût supplémentaire vous fait gagner de la flexibilité.

La toute première version d'Arc était un cas extrême de ce genre de lenteur à plusieurs niveaux, avec les avantages correspondants. Il s'agissait d'un interpréteur "métacirculaire" classique écrit sur Common Lisp, avec une certaine ressemblance familiale avec la fonction d'évaluation définie dans l'article original de Lisp de McCarthy. Le tout n'était que de quelques centaines de

lignes de code, donc c'était facile à comprendre et à changer. Le Common Lisp que nous avons utilisé, CLisp, s'exécute lui-même sur un interpréteur de code octet. Nous avions donc ici deux niveaux d'interprétation, l'un d'eux (le premier) étonnamment inefficace, et le langage était utilisable. À peine utilisable, je l'avoue, mais utilisable.

Écrire des logiciels en plusieurs couches est une technique puissante, même dans les applications. La programmation ascendante signifie l'écriture d'un programme sous la forme d'une série de couches, dont chacune sert de langage pour celle ci-dessus. Cette approche a tendance à produire des programmes plus petits et plus flexibles. C'est aussi la meilleure route vers ce Saint Graal, la réutilisabilité. Une langue est par définition réutilisable. Plus vous poussez votre application dans un langage permettant d'écrire ce type d'application, plus votre logiciel sera réutilisable.

D'une manière ou d'une autre, l'idée de réutilisabilité s'est attachée à la programmation orientée objet dans les années 1980, et aucune preuve du contraire ne semble être en mesure de la secouer librement. Mais bien que certains logiciels orientés objet soient réutilisables, ce qui le rend réutilisable, c'est sa position ascendante, pas son orientation objet. Considérez les bibliothèques : elles sont réutilisables parce qu'elles sont du langage, qu'elles soient écrites dans un style orienté objet ou non.

Je ne prédis pas la disparition de la programmation orientée objet, soit dit en passant. Bien que je ne pense pas qu'il ait grand-chose à offrir à de bons professionnels, sauf dans certains domaines spécialisés, il est irrésistible pour les grandes organisations. La programmation orientée objet offre un moyen durable d'écrire du code spaghetti. Il vous permet d'accréditer les programmes sous la forme d'une série de correctifs. Les grandes organisations ont toujours tendance à développer des logiciels de cette façon, et je m'attends à ce que cela soit aussi vrai dans cent ans qu'aujourd'hui.

Tant que nous parlons de l'avenir, nous ferions mieux de parler de calcul parallèle, parce que c'est là que cette idée semble vivre. À tout moment, cela semble toujours être quelque chose qui va se produire à l'avenir.

L'avenir le rattrapera-t-il un jour ? Les gens parlent du calcul parallèle comme de quelque chose d'imminent depuis au moins vingt ans, et cela n'a pas beaucoup affecté la pratique de la programmation jusqu'à présent. Ou n'est-ce pas ? Les concepteurs de puces doivent déjà y réfléchir, tout comme les gens qui essaient d'écrire des logiciels système sur des ordinateurs multi-CPU.

La vraie question est la suivante : jusqu'où ira le parallélisme dans l'échelle de l'abstraction ? Dans cent ans, cela affectera-t-il même les programmeurs d'applications ? Ou est-ce quelque chose auquel les rédacteurs compilateurs pensent, mais qui est généralement invisible dans le code source des applications ?

Une chose qui semble probable, c'est que la plupart des opportunités de parallélisme seront gaspillées. C'est un cas particulier de ma plus grande prédiction que la majeure partie de la puissance supplémentaire de l'ordinateur qui nous est donnée sera gaspillée. Je m'attends à ce que, comme pour la vitesse prodigieuse du matériel déséquilibré et, le parallélisme sera quelque chose qui sera disponible si vous le demandez explicitement, mais normalement pas utilisé. Cela implique que le genre de parallélisme que nous avons dans cent ans ne sera pas, sauf dans des applications spéciales, un parallélisme massif. Je m'attends à ce que pour les programmeurs ordinaires, il s'agira plutôt d'être en mesure de dériver des processus qui finissent tous par s'exécuter en parallèle.

Et cela, comme demander des implémentations spécifiques de structures de données, sera quelque chose que vous ferez assez tard dans la vie d'un programme, lorsque vous essayez de l'optimiser. Les versions 1 ignoreront normalement tous les avantages à tirer du calcul parallèle, tout comme elles ignoreront les avantages à tirer de représentations spécifiques de données.

Sauf dans les types spéciaux d'applications, le parallélisme n'imprégnera pas les programmes qui sont écrits dans cent ans. Ce serait une optimisation prématuée si c'était le cas.

Combien de langages de programmation y aura-t-il dans cent ans ? Il semble y avoir un grand nombre de nouveaux langages de programmation ces derniers temps. Une partie de la raison en est que le matériel plus rapide a permis aux programmeurs de faire différents compromis entre la vitesse et la

commodité, en fonction de l'application. S'il s'agit d'une véritable tendance, le matériel que nous aurons dans cent ans ne devrait que l'augmenter.

Et pourtant, il n'y a peut-être que quelques langages largement utilisés en cent ans. Une partie de la raison pour laquelle je dis cela est l'optimisme : il semble que, si vous faisiez un très bon travail, vous pourriez créer un langage idéal pour écrire une version lente 1, et pourtant, avec les bons conseils d'optimisation au compilateur, vous donneriez également un code rapide si nécessaire. Donc, puisque je suis optimiste, je vais prédire que malgré l'énorme écart qu'ils auront entre l'efficacité acceptable et l'efficacité maximale, les programmeurs dans cent ans auront des langages qui peuvent couvrir la majeure partie de celui-ci.

Au fur et à mesure que cet écart s'élargira, les profileurs deviendront de plus en plus importants. Peu d'attention est accordée au profilage maintenant. Beaucoup de gens semblent encore croire que la façon d'obtenir des applications rapides est d'écrire des compilateurs qui génèrent du code rapide. Au fur et à mesure que l'écart entre les performances acceptables et maximales s'élargit, il deviendra de plus en plus clair que la façon d'obtenir des applications rapides est d'avoir un bon guide de l'un à l'autre.

Quand je dis qu'il n'y a peut-être que quelques langages, je n'inclue pas les "petits langages" spécifiques au domaine. Je pense que de tels langages embarqués sont une excellente idée, et je m'attends à ce qu'ils prolifèrent. Mais je m'attends à ce qu'ils soient écrits comme des peaux assez fines pour que les utilisateurs puissent voir le langage général en dessous.

Qui concevra les langues du futur ? L'une des tendances les plus citées au cours des dix dernières années a été l'essor des langages open source comme Perl, Python et Ruby. La conception du langage est prise en charge par des hackers. Les résultats jusqu'à présent sont désordonnés, mais encourageants. Il y a des idées incroyablement nouvelles à Perl, par exemple. Beaucoup sont incroyablement mauvais, mais c'est toujours le cas des efforts ambitieux. À son taux de mutation actuel, Dieu sait en quoi Perl pourrait évoluer dans cent ans.

Il n'est pas vrai que ceux qui ne peuvent pas faire, enseignent (certains des meilleurs hackers que je connaisse sont des professeurs), mais il est vrai

qu'il y a beaucoup de choses que ceux qui enseignent ne peuvent pas faire. La recherche impose des restrictions contraignantes liées à la caste. Dans n'importe quel domaine académique, il y a des sujets sur lesquels il est acceptable de travailler et d'autres qui ne le sont pas. Malheureusement, la distinction entre les sujets acceptables et interdits est généralement basée sur la façon dont le travail semble intellectuel lorsqu'il est décrit dans les documents de recherche, plutôt que sur l'importance qu'il est pour obtenir de bons résultats. Le cas extrême est probablement la littérature ; les personnes qui étudient la littérature disent rarement quoi que ce soit qui serait de la moindre utilité pour ceux qui la produisent.

Bien que la situation soit meilleure dans les sciences, le chevauchement entre le genre de travail que vous êtes autorisé à faire et le genre de travail qui donne de bons langages est péniblement petit. (Olin Shivers s'est exprimé de manière très éloquente à ce sujet.) Par exemple, les types semblent être une source inépuisable d'articles de recherche, malgré le fait que le typage statique semble exclure les vrais macros - sans lesquelles, à mon avis, aucun langage ne vaut la peine d'être utilisé.

La tendance n'est pas seulement vers les langages développés en tant que projets open source plutôt que "recherche", mais aussi vers les langages conçus par les programmeurs d'applications qui ont besoin de les utiliser, plutôt que par les rédacteurs de compilateurs. Cela semble être une bonne tendance et je m'attends à ce qu'elle se poursuive.

Contrairement à la physique dans cent ans, qui est presque nécessairement impossible à prédire, il peut être possible en principe de concevoir un langage maintenant qui plairait aux utilisateurs dans cent ans.

Une façon de concevoir un langage est de simplement écrire le programme que vous aimeriez pouvoir écrire, qu'il y ait un compilateur qui peut le traduire ou du matériel qui peut l'exécuter. Lorsque vous faites cela, vous pouvez assumer des ressources illimitées. Il semble que nous devrions être en mesure d'imaginer des ressources illimitées aussi bien aujourd'hui qu'en cent ans.

Quel programme aimeraient-on écrire ? Quel que soit le moins de travail. Sauf que pas tout à fait : tout ce qui serait le moins fonctionnerait si vos idées sur la programmation n'étaient pas déjà influencées par les langages auxquels vous êtes actuellement habitué.es. Une telle influence peut être si omniprésente qu'il faut beaucoup d'efforts pour la surmonter. On pourrait penser qu'il serait évident pour des créatures aussi paresseuses que nous comment exprimer un programme avec le moins d'effort. En fait, nos idées sur ce qui est possible ont tendance à être si limitées par le langage que nous pensons que les formulations plus faciles de programmes semblent très surprenantes. C'est quelque chose que vous devez découvrir, pas quelque chose dans lequel vous vous enfoncez naturellement.

Une astuce utile ici est d'utiliser la longueur du programme comme une approximation de la quantité de travail qu'il est à écrire. Pas la longueur des caractères, bien sûr, mais la longueur des éléments syntaxiques distincts - fondamentalement, la taille de l'arbre d'analyse. Il n'est peut-être pas tout à fait vrai que le programme le plus court est le moins de travail à écrire, mais il est assez proche pour que vous soyez mieux de viser la cible solide de la brièveté que le flou, à proximité de l'un des moins de travail. Ensuite, l'algorithme de conception du langage devient : regardez un programme et demandez, y a-t-il un moyen plus court d'écrire ceci ?

En pratique, les programmes d'écriture dans une langue imaginaire centenaire fonctionneront à des degrés divers en fonction de votre proximité avec le noyau. Triez les routines que vous pouvez écrire maintenant. Mais il serait difficile de prédire maintenant quels types de bibliothèques pourraient être nécessaires dans cent ans. On peut supposer que de nombreuses bibliothèques seront pour des domaines qui n'existent même pas encore. Si SETI@home fonctionne, par exemple, nous aurons besoin de bibliothèques pour communiquer avec les extraterrestres. À moins bien sûr qu'ils ne soient suffisamment avancés pour qu'ils communiquent déjà en XML.

À l'autre extrême, je pense que vous pourriez être en mesure de concevoir le langage de base aujourd'hui. En fait, certains pourraient faire valoir qu'il a déjà été principalement conçu en 1958.

Si le langage centenaire était disponible aujourd'hui, voudrions-nous y programmer ? Une façon de répondre à cette question est de regarder en arrière. Si les langages de programmation actuels avaient été disponibles en 1960, quelqu'un aurait-il voulu les utiliser ?

D'une certaine manière, la réponse est non. Les langues d'aujourd'hui supposent une infrastructure qui n'existe pas en 1960. Par exemple, un langage dans lequel l'indentation est importante, comme Python, ne fonctionnerait pas très bien sur les terminaux d'imprimante. Mais en mettant de tels problèmes de côté - en supposant, par exemple, que les programmes étaient tout simplement écrits sur papier - les programmeurs des années 1960 auraient-ils aimé écrire des programmes dans les langues que nous utilisons maintenant ?

Je pense que c'est le cas. Certains des moins imaginatifs, qui avaient des faits d'art des premières langues intégrés dans leurs idées de ce qu'était un programme, auraient peut-être eu des problèmes. (Comment pouvez-vous manipuler les données sans faire de l'arithmétique des pointeurs ? Comment pouvez-vous mettre en œuvre des organigrammes sans gotos ?) Mais je pense que les programmeurs les plus intelligents n'auraient eu aucun mal à tirer le meilleur parti des langages d'aujourd'hui, s'ils les avaient eu.

Si nous avions le langage centenaire maintenant, cela ferait au moins un excellent pseudocode. Que diriez-vous de l'utiliser pour écrire des logiciels ? Étant donné que le langage centenaire devra générer du code rapide pour certaines applications, il pourrait vraisemblablement générer un code suffisamment efficace pour fonctionner de manière acceptable sur notre matériel. Nous devrons peut-être donner plus de conseils d'optimisation que les utilisateurs dans cent ans, mais cela pourrait toujours être une victoire nette.

Maintenant, nous avons deux idées qui, si vous les combinez, suggèrent des possibilités intéressantes : (1) le langage centenaire pourrait, en principe, être conçu aujourd'hui, et (2) un tel langage, s'il existait, pourrait être bon à programmer aujourd'hui. Quand vous voyez ces idées présentées comme ça, il est difficile de ne pas penser, pourquoi ne pas essayer d'écrire la langue centenaire maintenant ?

Lorsque vous travaillez sur la conception du langage, je pense qu'il est bon d'avoir une telle cible et de la garder consciemment à l'esprit. Lorsque vous apprenez à conduire, l'un des principes qu'ils vous enseignent est d'aligner la voiture non pas en alignant le capot avec les rayures peintes sur la route, mais en visant un moment donné dans la distance. Même si tout ce dont vous vous souciez, c'est de ce qui se passe dans les dix prochains pieds, c'est la bonne réponse. Je pense que nous devrions faire la même chose avec les langages de programmation

Chapitre 12

Battre les Moyennes

En 1995, Robert Morris et moi avons lancé une start-up appelée Viaweb. Notre plan était d'écrire des logiciels qui permettraient aux utilisateurs finaux de créer des boutiques en ligne. Ce qui était nouveau à propos de ce logiciel, à l'époque, c'est qu'il fonctionnait sur notre serveur, en utilisant des pages Web ordinaires comme interface.

Beaucoup de gens auraient pu avoir cette idée en même temps, bien sûr, mais pour autant que je sache, Viaweb a été la première application basée sur le Web. Cela nous a semblé une idée si nouvelle que nous avons nommé l'entreprise en son honneur : Viaweb, parce que notre logiciel fonctionnait via le Web, au lieu de s'exécuter sur votre ordinateur de bureau.

Une autre chose inhabituelle à propos de ce logiciel était qu'il était écrit principalement dans le langage de programmation appelé Lisp [1]. C'était l'une des premières grandes applications d'utilisateur final à être écrites en Lisp, qui jusque-là avait été utilisée principalement dans les universités et les laboratoires de recherche.

L'Arme Secrète

Eric Raymond a écrit un essai intitulé "How to Become a Hacker", et dans celui-ci, entre autres choses, il dit aux hackers potentiels quels langages ils devraient apprendre. Il suggère de commencer par Python et Java, car ils sont faciles à apprendre. Le hacker sérieux voudra également apprendre C, afin de bidouiller Unix, et Perl pour l'administration du système et les scripts CGI. Enfin, le hacker vraiment sérieux devrait envisager d'apprendre Lisp :

Lisp vaut la peine d'être appris pour l'expérience d'illumination profonde que vous aurez lorsque vous l'obtiendrez enfin ; cette expérience fera de vous un meilleur programmeur pour le reste de vos jours, même si vous n'utilisez jamais beaucoup Lisp lui-même.

C'est le même argument que vous avez tendance à entendre pour apprendre le latin. Cela ne vous donnera pas un emploi, sauf peut-être en tant

que professeur de classiques, mais cela améliorera votre esprit et fera de vous un meilleur écrivain dans les langues que vous voulez utiliser, comme l'anglais.

Mais attendez une minute. Cette métaphore ne va pas si loin. La raison pour laquelle le latin ne vous trouvera pas d'emploi est que personne ne le parle. Si vous écrivez en latin, personne ne peut vous comprendre. Mais Lisp est un langage informatique, et les ordinateurs parlent n'importe quelle langue que vous, le programmeur, leur dites.

Donc, si Lisp fait de vous un meilleur programmeur, comme il le dit, pourquoi ne voudriez-vous pas l'utiliser ? Si un peintre se voyait offrir un pinceau qui ferait de lui un meilleur peintre, il me semble qu'il voudrait l'utiliser dans toutes ses peintures, n'est-ce pas ? Je n'essaie pas de me moquer d'Eric Raymond ici. Dans l'ensemble, ses conseils sont bons. Ce qu'il dit à propos de Lisp est à peu près la sagesse conventionnelle. Mais il y a une contradiction dans la sagesse conventionnelle : Lisp fera de vous un meilleur programmeur, et pourtant vous ne l'utiliserez pas.

Pourquoi pas ? Après tout, les langages de programmation ne sont que des outils. Si Lisp donne vraiment de meilleurs programmes, vous devriez l'utiliser. Et si ce n'est pas le cas, alors qui en a besoin ?

Il ne s'agit pas seulement d'une question théorique. Le logiciel est une entreprise très compétitive, sujette à des monopoles naturels. Une entreprise qui obtient des logiciels plus rapidement et mieux rédigera, toutes les autres choses étant égales, mettra ses concurrents hors service. Et lorsque vous démarrez une start-up, vous le ressentez vivement. Les startups ont tendance à être une proposition de tout ou rien. Soit vous deviez riche, soit vous n'obtiendrez rien. Dans une start-up, si vous pariez sur la mauvaise technologie, vos concurrents vous écraseront.

Robert et moi connaissons bien Lisp, et nous ne voyions aucune raison de ne pas faire confiance à notre instinct et de l'utiliser. Nous savions que tout le monde écrivait son logiciel en C++ ou en Perl. Mais nous savions aussi que cela ne signifiait rien. Si vous choisissiez la technologie de cette façon, vous exécuteriez Windows. Lorsque vous choisissez la technologie, vous devez ignorer ce que font les autres et ne considérer que ce qui fonctionnera le mieux.



Avec Robert Morris, Viaweb, début 1996.

C'est particulièrement vrai dans une start-up. Dans une grande entreprise, vous pouvez faire ce que font toutes les autres grandes entreprises. Mais une startup ne peut pas faire ce que toutes les autres start-ups font. Je ne pense pas que beaucoup de gens s'en rendent compte, même dans les start-ups.

La grande entreprise moyenne croît à environ dix pour cent par an. Donc, si vous dirigez une grande entreprise et que vous faites tout comme le fait la grande entreprise moyenne, vous pouvez vous attendre à le faire aussi bien que la grande entreprise moyenne, c'est-à-dire à croître d'environ dix pour cent par an.

La même chose se produira si vous dirigez une start-up, bien sûr. Si vous faites tout comme le fait la start-up moyenne, vous devriez vous attendre à des performances moyennes. Le problème ici est que la performance moyenne signifie que vous allez faire faillite. Le taux de survie des start-ups est bien inférieur à cinquante pour cent. Donc, si vous dirigez une start-up, vous feriez mieux de faire quelque chose d'étrange. Sinon, vous aurez des ennuis.

En 1995, nous savions quelque chose que je ne pense pas que nos concurrents comprenaient, et peu le comprennent encore maintenant : lorsque vous écrivez des logiciels qui ne doivent s'exécuter que sur vos propres

serveurs, vous pouvez utiliser n'importe quel langage que vous voulez. Lorsque vous écrivez des logiciels de bureau, il y a un fort biais pour écrire des applications dans le même langage que le système d'exploitation. Il y a dix ans, la rédaction de demandes signifiait écrire des applications en C. Mais avec les logiciels basés sur le Web, en particulier lorsque vous avez le code source à la fois de la langue et du système d'exploitation, vous pouvez utiliser le langage que vous voulez.

Cette nouvelle liberté est cependant une épée à double tranchant. Maintenant que vous pouvez utiliser n'importe quel langage, vous devez réfléchir à celui à utiliser. Les entreprises qui essaient de prétendre que rien n'a changé risquent de constater que leurs concurrents ne le font pas.

Si vous pouviez utiliser n'importe quel langage, lequel utiliseriez-vous ? Nous avons choisi Lisp. D'une part, il était évident que le développement rapide serait important sur ce marché. Nous sommes tous partis de zéro, de sorte qu'une entreprise qui pourrait obtenir de nouvelles fonctionnalités avant que ses concurrents n'aient un grand avantage. Nous savions que Lisp était un très bon langage pour écrire des logiciels rapidement, et les applications basées sur le serveur amplifient l'effet du développement rapide, parce que vous pouvez publier des logiciels dès qu'ils sont terminés.

Si d'autres entreprises ne voulaient pas utiliser Lisp, tant mieux. Cela pourrait nous donner un avantage technologique, et nous avions besoin de toute l'aide que nous pouvions obtenir. Lorsque nous avons lancé Viaweb, nous n'avions aucune expérience dans les affaires. Nous ne savions rien sur le marketing, l'embauche de personnes, la collecte de fonds ou l'obtention de clients. Aucun d'entre nous n'avait jamais eu ce que vous appelleriez un vrai travail. La seule chose dans laquelle nous étions bons, c'était d'écrire des logiciels. Nous espérions que cela nous sauverait. Tout avantage que nous pourrions obtenir dans le département des logiciels, nous le ferions.

On pourrait donc dire que l'utilisation de Lisp était une expérimentation. Notre hypothèse était que si nous écrivions notre logiciel en Lisp, nous pourrions faire des fonctionnalités plus rapidement que nos concurrents, et aussi faire des choses dans notre logiciel qu'ils ne pouvaient pas faire. Et parce que Lisp était de si haut niveau, nous n'aurions pas besoin d'une grande équipe de

développement, donc nos coûts seraient plus bas. Si c'était le cas, nous pourrions offrir un meilleur produit pour moins d'argent, tout en profitant. Nous finirions par obtenir tous les utilisateurs, et nos concurrents n'en obtiendraient aucun, et finiraient par fermer leurs portes. C'est ce que nous espérions qu'il se passerait, de toute façon.

Quels ont été les résultats de cette expérience ? Étonnamment, ça a marché. Nous avons finalement eu beaucoup de concurrents, environ vingt à trente d'entre eux, mais aucun de leurs logiciels ne pouvait rivaliser avec le nôtre. Nous avions un constructeur de boutique en ligne wysiwyg qui s'est exécuté sur le serveur et qui ressemblait pourtant à une application de bureau. Nos concurrents avaient des scripts CGI. Et nous étions toujours loin devant eux en ce qui concerne les caractéristiques. Parfois, dans le désespoir, les concurrents essayaient d'introduire des fonctionnalités que nous n'avions pas. Mais avec Lisp, notre cycle de développement était si rapide que nous pouvions parfois dupliquer une nouvelle fonctionnalité dans un délai d'un jour ou deux après qu'un concurrent l'annonçait dans un communiqué de presse. Au moment où les journalistes couvrant le communiqué de presse ont pu nous appeler, nous aurions aussi la nouvelle fonctionnalité.

Il a dû sembler à nos concurrents que nous avions une sorte d'arme secrète - que nous décodions leur trafic Enigma ou quelque chose comme ça. En fait, nous avions une arme secrète, mais c'était plus simple qu'ils ne le pensent. Personne ne nous a divulgué des nouvelles de leurs caractéristiques. Nous avons juste été en mesure de développer des logiciels plus rapidement que quiconque ne le pensait possible.

Quand j'avais environ neuf ans, il m'est arrivé de trouver une copie de *The Day of the Jackal*, de Frederick Forsyth. Le personnage principal est un assassin qui est engagé pour tuer le président de la France. L'assassin doit passer devant la police pour se rendre dans un appartement qui surplombe la route du président. Il passe juste à côté d'eux, déguisé en vieil homme sur des béquilles, et ils ne le soupçonnent jamais.

Notre arme secrète était similaire. Nous avons écrit notre logiciel dans un langage d'IA bizarre, avec une syntaxe bizarre pleine de parenthèses. Pendant des années, cela m'avait ennuyé d'entendre Lisp décrire de cette façon. Mais

maintenant, cela a fonctionné à notre avantage. En affaires, il n'y a rien de plus précieux qu'un avantage technique que vos concurrents ne comprennent pas. Dans les affaires, comme dans la guerre, la surprise vaut autant que la force.

Et donc, je suis un peu gêné de dire que je n'ai jamais rien dit publiquement à propos de Lisp pendant que nous travaillions sur Viaweb. Nous n'en avons jamais parlé à la presse, et si vous cherchez Lisp sur notre site web, tout ce que vous trouverez, ce sont les titres de deux livres dans ma biographie. Ce n'était pas un hasard. Une start-up devrait donner à ses concurrents le moins d'informations possible. S'ils ne savaient pas quelle langue notre logiciel était écrit, ou s'en fichaient, je voulais le laisser comme ça [2].

Les personnes qui comprenaient le mieux notre technologie étaient les clients. Ils ne se souciaient pas non plus de la langue dans laquelle Viaweb était écrit, mais ils ont remarqué que cela fonctionnait vraiment bien. Cela leur a permis de construire de superbes boutiques en ligne en littéralement quelques minutes. Et donc, par bouche à oreille principalement, nous avons de plus en plus d'utilisateurs. À la fin de 1996, nous avions environ 70 magasins en ligne. À la fin de 1997, nous en avions 500. Six mois plus tard, lorsque Yahoo nous a achetés, nous avions 1070 utilisateurs. Aujourd'hui, en tant que Yahoo Store, ce logiciel continue de dominer son marché. C'est l'une des pièces les plus rentables de Yahoo, et les magasins construits avec elle sont la base de Yahoo Shopping. J'ai quitté Yahoo en 1999, donc je ne sais pas exactement combien d'utilisateurs ils ont maintenant, mais la dernière fois que j'ai entendu dire qu'il y en avait plus de 20 000.

Le Paradoxe de Blub

Qu'y a-t-il de si génial à propos de Lisp ? Et si Lisp est si génial, pourquoi tout le monde ne l'utilise-t-il pas ? Celles-ci ressemblent à des questions rhétoriques, mais en fait, elles ont des réponses simples. Lisp est si grand non pas à cause d'une qualité magique visible uniquement par les dévots, mais parce que c'est tout simplement le langage le plus puissant disponible. Et la raison pour laquelle tout le monde ne l'utilise pas est que les langages de programmation ne sont pas seulement des technologies, mais aussi des habitudes d'esprit, et rien ne change plus lentement. Bien sûr, ces deux réponses doivent être expliquées.

Je vais commencer par une déclaration étonnamment controversée : les langages de programmation varient en puissance.

Peu de gens contesteraient, du moins, le fait que les langages de haut niveau sont plus puissants que le langage machine. La plupart des programmeurs d'aujourd'hui seraient d'accord pour dire que vous ne voulez pas, normalement, programmer en machine. Au lieu de cela, vous devriez programmer dans un langage de haut niveau et demander à un compilateur de le traduire en langage machine pour vous. Cette idée est même intégrée au matériel maintenant : depuis les années 1980, les ensembles d'instructions ont été conçus pour les compilateurs plutôt que pour les programmeurs humains.

Tout le monde sait que c'est une erreur d'écrire tout votre programme à la main en langage machine. Ce qui est moins souvent compris, c'est qu'il y a un principe plus général ici : si vous avez le choix entre plusieurs langues, c'est, toutes choses étant égales par ailleurs, une erreur de programmer dans autre chose que la plus puissante [3].

Il y a de nombreuses exceptions à cette règle. Si vous écrivez un programme qui doit travailler en étroite collaboration avec un programme écrit dans un certain langage, ce pourrait être une bonne idée d'écrire le nouveau programme dans le même langage. Si vous écrivez un programme qui n'a qu'à faire quelque chose de simple, comme le craquement des nombres ou la manipulation de bits, vous pouvez aussi bien utiliser un langage moins abstrait, d'autant plus qu'il peut être légèrement plus rapide. Et si vous écrivez un programme court et jetable, vous feriez peut-être mieux d'utiliser n'importe quel langage qui a les meilleures bibliothèques pour la tâche. Mais en général, pour les logiciels d'application, vous voulez utiliser le langage le plus puissant (raisonnablement efficace) que vous puissiez obtenir, et utiliser quoi que ce soit d'autre est une erreur, exactement du même type, bien que peut-être dans une moindre mesure, que la programmation en langage machine.

Vous pouvez *voir* que le langage machine est de très bas niveau. Mais, au moins comme une sorte de convention sociale, les langages de haut niveau sont souvent tous traités comme équivalents. Ils ne le sont pas. Techniquement, le terme « langage de haut niveau » ne signifie rien de très précis. Il n'y a pas de

ligne de démarcation avec les langages machine d'un côté et tous les langages de haut niveau de l'autre. Les langages s'inscrivent le long d'un continuum d'abstraction [4], des langages les plus puissants jusqu'aux langages des machines, qui varient eux-mêmes en pouvoir.

Considérez Cobol. Cobol est un langage de haut niveau, en ce sens qu'il est compilé en langage machine. Quelqu'un pourrait-il sérieusement faire valoir que Cobol est équivalent en puissance à, par exemple, Python ? C'est probablement plus proche du langage machine que de Python.

Où que diriez-vous de Perl 4 ? Entre Perl 4 et Perl 5, des fermetures lexicales ont été ajoutées à la langue. La plupart des pirates Perl seraient d'accord pour dire que Perl 5 est plus puissant que Perl 4. Mais une fois que vous avez admis cela, vous avez admis qu'une langue de haut niveau peut être plus puissante qu'une autre. Et il s'ensuit inexorablement que, sauf dans des cas particuliers, vous devriez utiliser le plus puissant que vous puissiez obtenir.

Cependant, cette idée est rarement suivie à sa conclusion. Après un certain âge, les programmeurs changent rarement de langage volontairement. Quel que soit le langage auquel les gens sont habitués, ils ont tendance à être assez bons.

Les programmeurs s'attachent beaucoup à leurs langages préférés, et je ne veux blesser personne, alors pour expliquer ce point, je vais utiliser un langage hypothétique appelé Blub. Blub tombe en plein milieu du continuum de l'abstraction. Ce n'est pas le langage le plus puissant, mais il est plus puissant que le Cobol ou le langage machine.

Et en fait, notre hypothétique programmeur Blub n'utilisera ni l'un ni l'autre. Bien sûr, il ne programmerait pas en langage machine. C'est à cela que servent les compilateurs. Et quant à Cobol, il ne sait pas comment quelqu'un peut en faire quoi que ce soit. Il n'a même pas de x (fonction Blub de votre choix).

Tant que notre hypothétique programmeur Blub regarde vers le bas le continuum de puissance, il sait qu'il regarde vers le bas. Les langages moins puissants que Blub sont évidemment moins puissants, car il leur manque une

fonctionnalité à laquelle il est habitué. Mais quand notre hypothétique programmeur Blub regarde dans l'autre sens, vers le haut du continuum de puissance, il ne se rend pas compte qu'il regarde vers le haut. Ce qu'il voit ne sont que des langages bizarres. Il les considère probablement comme équivalant au pouvoir à Blub, mais avec toutes ces autres choses poilues jetées aussi. Blub est assez bon pour lui, parce qu'il pense à Blub.

Cependant, lorsque nous passons au point de vue d'un programmeur utilisant l'un des langages plus haut dans le continuum de puissance, nous constatons qu'il regarde à son tour Blub. Comment pouvez-vous faire quelque chose dans Blub ? Il n'a même pas de y.

Par induction, les seuls programmeurs en mesure de voir toutes les différences de puissance entre les différents langages sont ceux qui comprennent le plus puissant. (C'est probablement ce qu'Eric Raymond voulait dire à propos de Lisp qui fait de vous un meilleur programmeur.) Vous ne pouvez pas faire confiance aux opinions des autres, à cause du paradoxe Blub : ils sont satisfaits du langage qu'ils utilisent, parce que cela dicte la façon dont ils pensent aux programmes.

Je le sais d'après ma propre expérience, en tant qu'élève du lycée, écrivant des programmes dans Basic. Ce langage ne prenait même pas en charge la récursivité. Il est difficile d'imaginer écrire des programmes sans utiliser la récursivité, mais je ne l'ai pas manqué à l'époque. J'ai pensé dans Basic. Et j'étais un génie. Maître de tout ce que j'ai interrogé.

Les cinq langages qu'Eric Raymond recommande aux hackers tombent à différents points du continuum du pouvoir. L'endroit où ils se situent les uns par rapport aux autres est un sujet sensible. Ce que je vais dire, c'est que je pense que Lisp est au sommet. Et pour étayer cette affirmation, je vais vous parler de l'une des choses qui me manquent lorsque je regarde les quatre autres langages. Comment pouvez-vous faire quoi que ce soit en eux, je pense, sans macros ? [5]

De nombreux langages ont ce qu'on appelle une macro. Mais les macros Lisp sont uniques. Et croyez-le ou non, ce qu'ils font est réappuyé entre parenthèses. Les concepteurs de Lisp n'ont pas mis toutes ces parenthèses dans le langage juste pour être différents. Pour le programmeur Blub, le code Lisp a

l'air bizarre. Mais ces parenthèses sont là pour une raison. Ils sont la preuve extérieure d'une différence fondamentale entre le lisp et les autres langages.

Le code Lisp est fabriqué à partir d'objets de données Lisp. Et pas dans le sens trivial que les fichiers sources contiennent des caractères, et les chaînes sont l'un des types de données pris en charge par le langage. Le code Lisp, après avoir été lu par l'analyseur, est composé de structures de données que vous pouvez parcourir.

Si vous comprenez comment fonctionnent les compilateurs, ce qui se passe vraiment, ce n'est pas tant que Lisp a une syntaxe étrange que Lisp n'a pas de syntaxe. Vous écrivez des programmes dans les arbres d'analyse qui sont générés dans le compilateur lorsque d'autres langages sont analysés. Mais ces arbres d'analyse sont entièrement accessibles à vos programmes. Vous pouvez écrire des programmes qui les manipulent. Dans Lisp, ces programmes sont appelés macros. Ce sont des programmes qui écrivent des programmes.

Des programmes qui écrivent des programmes ? Quand voudriez-vous faire ça ? Pas très souvent, si vous pensez à Cobol. Tout le temps, si vous pensez à Lisp. Ce serait pratique ici si je pouvais donner un exemple d'une macro puissante, et dire, là ! Que diriez-vous de ça ? Mais si je le faisais, cela ressemblerait à du charabia pour quelqu'un qui ne connaît pas Lisp ; il n'y a pas de place ici pour expliquer tout ce que vous auriez besoin de savoir pour comprendre ce que cela signifiait. Dans Ansi Common Lisp, j'ai essayé de faire avancer les choses aussi vite que possible, et même ainsi, je n'ai pas atteint les macros avant la page 160.

Mais je pense que je peux donner une sorte d'argument qui pourrait être convaincant. Le code source de l'éditeur Viaweb était probablement d'environ 20 à 25 % de macros. Les macros sont plus difficiles à écrire que les fonctions Lisp ordinaires, et c'est un mauvais style de les utiliser lorsqu'elles ne sont pas nécessaires. Donc, chaque macro de ce code est là parce qu'elle doit l'être. Cela signifie qu'au moins 20 à 25 % du code de ce programme fait des choses que vous ne pouvez pas facilement faire dans une autre langue. Quelle que soit la sceptique que le programmeur Blub puisse être au sujet de mes revendications pour les pouvoirs mystérieux de Lisp, cela devrait le rendre curieux. Nous n'écrivions pas ce code pour notre propre amusement. Nous étions une petite

start-up, programmant aussi dur que possible afin de mettre des barrières techniques entre nous et nos concurrents.

Une personne suspecte pourrait commencer à se demander s'il y avait une corrélation ici. Une grande partie de notre code faisait des choses qui sont difficiles à faire dans d'autres langues. Le logiciel qui en a résulté a fait des choses que le logiciel de nos concurrents ne pouvait pas faire. Peut-être qu'il y avait une sorte de connexion. Je vous encourage à suivre ce fil de discussion. Il y a peut-être plus à ce vieil homme qui traîne sur ses béquilles qu'il n'y paraît.

Aïkido pour les Startups

Mais je ne m'attends pas à convaincre quelqu'un (de plus de 25 ans) de sortir et d'apprendre le Lisp. Mon but ici n'est pas de changer l'avis de qui que ce soit, mais de rassurer les gens déjà intéressés par l'utilisation de Lisp - les gens qui savent que Lisp est un langage puissant, mais qui s'inquiètent parce qu'il n'est pas largement utilisé. Dans une situation concurrentielle, c'est un avantage. La puissance de Lisp est multipliée par le fait que vos concurrents ne l'obtiennent pas.

Si vous pensez à utiliser Lisp dans une start-up, vous ne devriez pas vous inquiéter qu'il ne soit pas largement compris. Vous devriez espérer que cela reste ainsi. Et c'est probable. C'est la nature des langages de programmation qui satisfait la plupart des gens de ce qu'ils utilisent actuellement. Le matériel informatique change tellement plus rapidement que les habitudes personnelles que la pratique de la programmation est généralement de dix à vingt ans derrière le processeur. Dans des endroits comme le MIT, ils écrivaient des programmes dans des langages de haut niveau au début des années 1960, mais de nombreuses entreprises ont continué à écrire du code en langage machine jusque dans les années 1980. Je parie que beaucoup de gens ont continué à écrire le langage machine jusqu'à ce que le processeur, comme un barman désireux de fermer et de rentrer chez lui, les expulse finalement en passant à un ensemble d'instructions RISC.

Habituellement, la technologie change rapidement. Mais les langages de programmation sont différents : les langages de programmation ne sont pas seulement la technologie, mais ce à quoi les programmeurs pensent. Ils sont à

moitié technologiques et à moitié religieux [6]. Et donc le langage médian, c'est-à-dire quel que soit le langage utilisé par le programmeur médian, se déplace aussi lentement qu'un iceberg. La collecte des ordures, introduite par Lisp vers 1960, est maintenant largement considérée comme une bonne chose. La dactylographie dynamique, idem, gagne en popularité. Les fermetures lexicales, introduites par Lisp au début des années 1960, sont maintenant, à peine, sur l'écran radar. Les macros, introduites par Lisp au milieu des années 1960, sont encore terra incognita.

De toute évidence, le langage médian a un élan énorme. Je ne propose pas que vous puissiez combattre cette force puissante. Ce que je propose, c'est exactement le contraire : que, comme un pratiquant de l'Aïkido, vous pouvez l'utiliser contre vos adversaires.

Si vous travaillez pour une grande entreprise, ce n'est peut-être pas facile. Vous aurez du mal à convaincre le patron aux cheveux pointus de vous laisser construire des choses dans Lisp, alors qu'il vient de lire dans le journal qu'un autre langage est prêt, comme Ada l'était il y a vingt ans, à prendre le contrôle du monde. Mais si vous travaillez pour une start-up qui n'a pas encore de patrons aux cheveux pointus, vous pouvez, comme nous, transformer le paradoxe Blub à votre avantage : vous pouvez utiliser une technologie que vos concurrents, collés de manière immuable au langage médian, ne seront jamais en mesure de faire correspondre.

Si jamais vous vous retrouvez à travailler pour une start-up, voici un conseil pratique pour évaluer les concurrents. Lisez leurs offres d'emploi. Tout le reste sur leur site peut être des photos de stock ou l'équivalent en prose, mais les offres d'emploi doivent être spécifiques à ce qu'ils veulent, sinon ils auront les mauvais candidats.

Au cours des années où nous avons travaillé sur Viaweb, j'ai lu beaucoup de descriptions de travail. Un nouveau concurrent semblait émerger du travail du bois tous les mois environ. La première chose que je ferais, après avoir vérifié s'ils avaient une démo en ligne en direct, était de regarder leurs offres d'emploi. Après quelques années de cela, je pouvais dire quelles entreprises s'inquiéter et lesquelles ne pas s'inquiéter. Plus les décryptages informatiques avaient une saveur informatique, moins l'entreprise était dangereuse. Le type le

plus sûr était celui qui voulait une expérience Oracle. Vous n'avez jamais eu à vous en soucier. Vous étiez également en sécurité s'ils disaient qu'ils voulaient des développeurs C++ ou Java. S'ils voulaient des programmeurs Perl ou Python, ce serait un peu effrayant - cela commence à ressembler à une entreprise où le côté technique, au moins, est géré par de vrais hackers. Si j'avais déjà vu une offre d'emploi à la recherche de hackers Lisp, j'aurais été vraiment inquiet.

Chapitre 13

La Revanche des Nerds

Dans le secteur des logiciels, il y a une lutte en cours, entre les universitaires à la tête pointue et une autre force tout aussi formidable, les patrons aux cheveux pointus. Je crois que tout le monde sait qui est le patron aux cheveux pointues [1]. Je pense que la plupart des gens dans le monde de la technologie reconnaissent non seulement ce personnage de dessin animé, mais connaissent la personne réelle dans leur entreprise sur laquelle il est calqué.

Le patron aux cheveux pointus combine miraculeusement deux qualités qui sont communes à elles-mêmes, mais rarement vues ensemble : (a) il ne sait rien du tout de la technologie, et (b) il a des opinions très fortes à ce sujet.

Supposons, par exemple, que vous ayez besoin d'écrire un logiciel. Le patron aux cheveux pointus n'a aucune idée de la façon dont ce logiciel doit fonctionner et ne peut pas distinguer un langage de programmation d'un autre, et pourtant il sait dans quel langage vous devriez l'écrire. Exactement. Il pense que vous devriez l'écrire en Java.

Pourquoi pense-t-il cela ? Jetons un coup d'œil à l'intérieur du cerveau du patron aux cheveux pointus. Ce qu'il pense, c'est quelque chose comme ça. Java est une norme. Je sais que ça doit l'être, parce que je lis tout le temps à ce sujet dans la presse. Comme il s'agit d'une norme, je n'aurai pas de problèmes pour l'utiliser. Et cela signifie également qu'il y aura toujours beaucoup de programmeurs Java, donc si ceux qui travaillent pour moi démissionnent maintenant, comme le font mystérieusement toujours les programmeurs qui travaillent pour moi, je peux facilement les remplacer.

Eh bien, cela ne semble pas si déraisonnable. Mais tout est basé sur une hypothèse tacite, et cette hypothèse s'avère fausse. Le patron aux cheveux pointus pense que tous les langages de programmation sont à peu près équivalents. Si c'était vrai, il serait sur la bonne voie. Si les langages sont tous équivalents, bien sûr, utilisez le langage que tout le monde utilise.

Mais tous les langages ne sont pas équivalents, et je pense que je peux vous le prouver sans même entrer dans les différences entre eux. Si vous aviez

demandé au patron aux cheveux pointu en 1992 dans quel langage le langage doux devrait être écrit, il aurait répondu avec aussi peu d'hésitation qu'aujourd'hui. Le logiciel doit être écrit en C++. Mais si les langues sont toutes équivalentes, pourquoi l'opinion du patron aux cheveux pointus devrait-elle jamais changer ? En fait, pourquoi les développeurs de Java auraient-ils même pris la peine de créer un nouveau langage ?

Vraisemblablement, si vous créez un nouveau langage, c'est parce que vous pensez qu'il est meilleur d'une manière ou d'une autre que ce que les gens avaient déjà. Et en fait, Gosling indique clairement dans le premier livre blanc sur Java, que ce langage a été conçu pour résoudre certains problèmes avec C++. Alors voilà : les langages ne sont pas tous équivalents. Si vous suivez la piste à travers le cerveau du patron aux cheveux pointus jusqu'à Java, puis à travers l'histoire de Java jusqu'à ses origines, vous finissez par avoir une idée qui contredit l'hypothèse avec laquelle vous avez commencé.

Alors, qui a raison ? James Gosling, ou le patron aux cheveux pointus ? Il n'est pas surprenant que Gosling ait raison. Certains langages sont meilleurs, pour certains problèmes, que d'autres. Et vous savez, cela soulève des questions intéressantes. Java a été conçu pour être meilleur, pour certains problèmes, que C++. Quels problèmes ? Quand Java est-il meilleur et quand est-ce que C++ ? Y a-t-il des situations où d'autres langages sont meilleurs que l'un ou l'autre ?

Une fois que vous avez commencé à considérer cette question, vous avez ouvert une vraie canne de vers. Si le patron aux cheveux pointus devait penser au problème dans toute sa complexité, cela lui ferait exploser la tête. Tant qu'il considère tous les langages comme équivalents, tout ce qu'il a à faire est de choisir celui qui semble avoir le plus d'élan, et comme c'est plus une question de mode que de technologie, même lui peut probablement obtenir la bonne réponse. Mais si les langages varient, il doit soudainement résoudre deux équations simultanées, en essayant de trouver un équilibre optimal entre deux choses dont il ne sait rien : l'adéquation relative de la vingtaine de langages principaux pour le problème qu'il doit résoudre, et les chances de trouver des programmeurs, des bibliothèques, etc. pour chacun. Si c'est ce qui se trouve de l'autre côté de la porte, il n'est pas surprenant que le patron aux cheveux pointus ne veuille pas l'ouvrir.

L'inconvénient de croire que tous les langages de programmation sont équivalents est que ce n'est pas vrai. Mais l'avantage est que cela rend votre vie beaucoup plus simple. Et je pense que c'est la principale raison pour laquelle l'idée est si répandue. C'est une idée *confortable*.

Nous savons que Java doit être assez bon, parce que c'est le nouveau langage de programmation cool. Ou est-ce ou est-ce le cas ? Si vous regardez le monde des langages de programmation de loin, il semble que Java soit la dernière chose. (De loin, tout ce que vous pouvez voir, c'est le grand panneau d'affichage clignotant payé par Sun.) Mais si vous regardez ce monde de près, vous trouverez qu'il y a des degrés de fraîcheur. Au sein de la sous-culture hacker, il existe un autre langage appelé Perl qui est considéré comme beaucoup plus cool que Java. Slashdot, par exemple, est généré par Perl. Je ne pense pas que vous trouveriez ces gars qui utilisent Java Server Pages. Mais il existe un autre langage plus récent, appelé Python, dont les utilisateurs ont tendance à regarder vers le bas sur Perl, et un autre appelé Ruby que certains considèrent comme l'héritier apparent de Python.

Si vous regardez ces langages dans l'ordre, Java, Perl, Python, Ruby, vous remarquez un modèle intéressant. Au moins, vous remarquez ce modèle si vous êtes un hacker Lisp. Chacun ressemble progressivement plus à Lisp. Python copie même des fonctionnalités que de nombreux hackers Lisp considèrent comme des erreurs. Et si vous aviez montré aux gens Ruby en 1975 et l'avoir décrit comme un dialecte de Lisp avec une syntaxe, personne ne se serait disputé avec vous. Les langages de programmation ont presque rattrapé 1958.

Rattraper les mathématiques

Ce que je veux dire, c'est que Lisp a été découvert pour la première fois par John McCarthy en 1958, et que les langages de programmation populaires ne font que rattraper les idées qu'il a développées à l'époque.

Maintenant, comment cela a-t-il pu être vrai ? La technologie informatique n'est-elle pas quelque chose qui change très rapidement ? En 1958, les ordinateurs étaient des mastodontes de la taille d'un réfrigérateur avec la puissance de traitement d'une montre-bracelet [2]. Comment une technologie

aussi ancienne pourrait-elle même être pertinente, et encore moins supérieure aux derniers développements ?

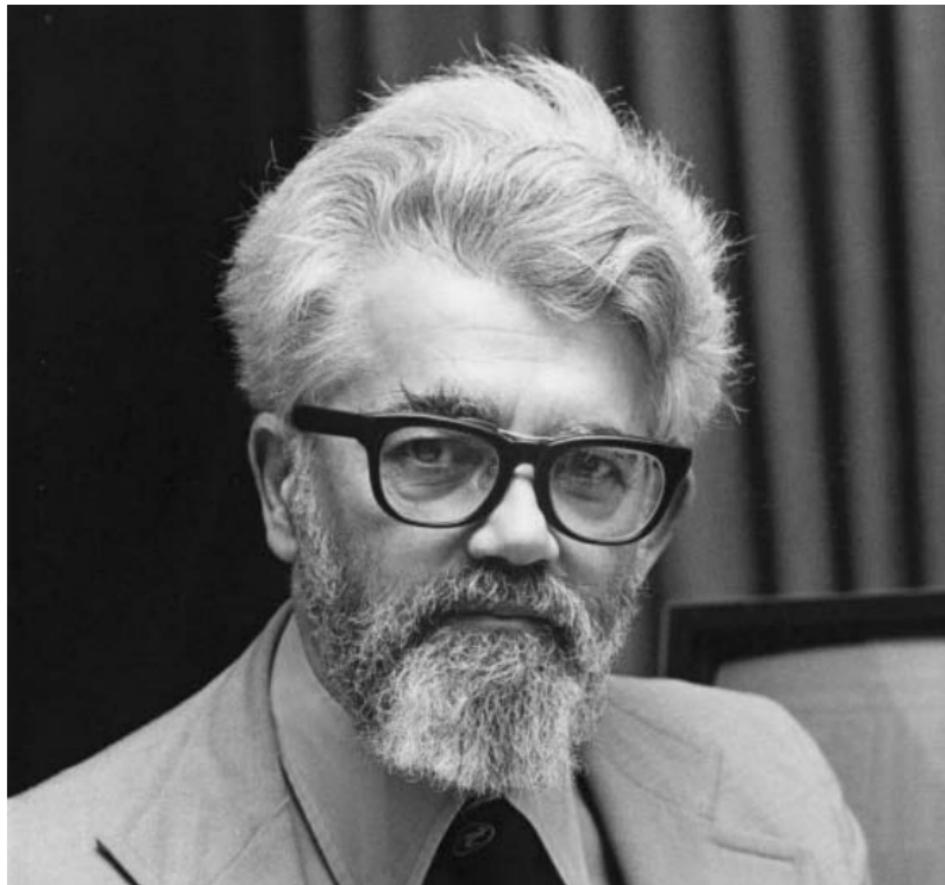


IBM 704, Lawrence Livermore, 1956.

Je vais vous dire comment. C'est parce que Lisp n'a pas vraiment été conçu pour être un langage de programmation, du moins pas dans le sens que nous voulons dire aujourd'hui. Ce que nous entendons par langage de programmation est quelque chose que nous utilisons pour dire à un ordinateur ce qu'il doit faire. McCarthy a finalement eu l'intention de développer un langage de programmation dans ce sens, mais le Lisp avec lequel nous nous sommes retrouvés était basé sur quelque chose de distinct qu'il a fait comme un exercice théorique - un effort pour définir une alternative plus pratique à la machine de Turing. Comme McCarthy l'a dit plus tard :

Une autre façon de montrer que Lisp était plus propre que les machines de Turing était d'écrire une fonction Lisp universelle et de montrer qu'elle est plus brève et plus compréhensible que la description d'une machine de Turing universelle. Il s'agissait de l'évaluation de la fonction Lisp. . . , qui calcule la valeur d'une expression Lisp. . . Pour écrire eval, il fallait inventer une notation représentant les fonctions Lisp

en tant que données Lisp, et une telle notation a été conçue aux fins de l'article sans penser qu'elle serait utilisée pour exprimer les programmes Lisp dans la pratique.



Alpha nerd : John McCarthy.

Mais à la fin de 1958, Steve Russell [3], l'un des étudiants diplômés de McCarthy, a examiné cette définition de l'évaluation et s'est rendu compte que s'il la traduisait en langage machine, le résultat serait un interprète Lisp.

C'était une grande surprise à l'époque. Voici ce que McCarthy a dit à ce sujet plus tard :

Steve Russell a dit, regardez, pourquoi ne pas programmer cette *évaluation*..., et je lui ai dit, ho, ho, vous confondez la théorie avec la pratique, cette évaluation est destinée à la lecture, pas à l'informatique. Mais il est allé de l'avant et l'a fait. C'est-à-dire qu'il a compilé *l'évaluation* dans mon article dans le code machine [IBM] 704, corrigeant les bugs, puis l'a annoncé comme un interprète Lisp, ce qui était

certainement le cas. Donc, à ce moment-là, Lisp avait essentiellement la forme qu'il a aujourd'hui. . . .

Soudain, en quelques semaines, McCarthy a trouvé son exercice théorique transformé en un véritable langage de programmation - et plus puissant qu'il ne l'avait prévu.

Donc, la brève explication de la raison pour laquelle ce langage des années 1950 n'est pas obsolète est qu'il ne s'agissait pas de technologie, mais de mathématiques, et que les mathématiques ne deviennent pas obsolètes. La bonne chose à comparer Lisp n'est pas le matériel des années 1950, mais l'algorithme Quicksort, qui a été découvert en 1960 et qui est toujours le tri polyvalent le plus rapide.

Il y a un autre langage qui subsiste encore depuis les années 1950, Fortran, et il représente l'approche opposée de la conception du langage. Lisp était un morceau de théorie qui s'est transformé de manière inattendue en langage de programmation. Fortran a été développé intentionnellement en tant que langage de programmation, mais ce que nous considérons maintenant comme un langage de très bas niveau.

Fortran I, le langage qui a été développé en 1956, était un animal très différent de l'actuel Fortran. Fortran, j'étais à peu près en langage assembleur avec les mathématiques. D'une certaine manière, il était moins puissant que les langages d'assemblage plus récents ; il n'y avait pas de sous-routines, par exemple, seulement des branches. Aujourd'hui, Fortran est maintenant sans doute plus proche de Lisp que de Fortran I.

Lisp et Fortran étaient les troncs de deux arbres évolutifs distincts, l'un enraciné dans les mathématiques et l'autre dans l'architecture des machines. Ces deux arbres convergent depuis lors. Lisp a commencé par être puissant et, au cours des vingt années suivantes, il s'est accéléré. Les langages dits traditionnels ont commencé rapidement, et au cours des quarante années suivantes, ils sont devenus progressivement plus puissants, jusqu'à présent, les plus avancées d'entre elles sont assez proches de Lisp. Proche, mais il leur manque encore quelque chose.

Qu'est-ce qui a rendu Lisp différent

Lorsqu'il a été développé pour la première fois, Lisp a incarné neuf nouvelles idées. Certains d'entre eux que nous tenons maintenant pour acquis, d'autres ne sont vus que dans des langages plus avancés, et deux sont toujours uniques à Lisp. Les neuf idées sont, dans l'ordre de leur adoption par le grand public,

1. Conditionnels. Un conditionnel est une construction “if-then-else”. Nous les tenons pour acquis maintenant, mais Fortran, je ne les avais pas. Il n'avait qu'un goto conditionnel étroitement basé sur l'instruction de la machine sous-jacente.
2. Un type de fonction. Dans Lisp, les fonctions sont un type de données tout comme des entiers ou des chaînes. Ils ont une représentation littérale, peuvent être stockés dans des variables, peuvent être passés en tant qu'arguments, et ainsi de suite.
3. Récursion. Lisp a été le premier langage de haut niveau à prendre en charge les fonctions récursives [4].
4. Saisie dynamique. Dans Lisp, toutes les variables sont effectivement des pointeurs. Les valeurs sont ce qui a des types, pas des variables, et attribuer des valeurs à des variables signifie copier des pointeurs, pas ce qu'elles pointent.
5. Collecte des déchets.
6. Programmes composés d'expressions. Les programmes Lips sont des arbres d'expression, dont chacun renvoie une valeur. Cela contraste avec le Fortran et la plupart des langues suivantes, qui font la distinction entre les expressions et les déclarations.

Cette distinction était naturelle chez Fortran I parce que vous ne pouviez pas imbriquer les déclarations. Donc, alors que vous aviez besoin d'expressions pour que les mathématiques fonctionnent, il ne faisait aucun

intérêt à faire en sorte que quoi que ce soit d'autre renvoie une valeur, parce qu'il ne pouvait y avoir rien qui l'attendait.

Cette limitation a disparu avec l'arrivée des langages structurés en bloc, mais à ce moment-là, il était trop tard. La distinction entre les expressions et les déclarations était engrainée. Il s'est propagé de Fortran à Algol, puis à leurs deux descendants.

7. Un type de symbole. Les symboles sont en fait des pointeurs vers des chaînes de caractères stockées dans une table de hachage. Vous pouvez donc tester l'égalité en comparant un pointeur, au lieu de comparer chaque caractère.
8. Une notation pour le code utilisant des arbres de symboles et de constantes.
9. Toute la langue là-bas tout le temps. Il n'y a pas de réelle différence entre le temps de lecture, le temps de compilation et l'exécution. Vous pouvez compiler ou exécuter du code pendant la lecture, lire ou exécuter du code pendant la compilation, et lire ou compiler du code au moment de l'exécution.

L'exécution de code au moment de la lecture permet aux utilisateurs de reprogrammer la syntaxe de Lisp ; l'exécution du code au moment de la compilation est la base des macros ; la compilation à l'exécution est la base de l'utilisation de Lisp comme langage d'extension dans des programmes comme Emacs ; et la lecture à l'exécution permet aux programmes de communiquer en utilisant des *s-expressions*, une idée récemment réinventée comme XML [5].

Lorsque Lisp est apparu pour la première fois, ces idées étaient loin de la pratique de programmation ordinaire, qui était dictée en grande partie par le matériel disponible à la fin des années 1950. Au fil du temps, le langage par défaut, incarné dans une succession de langues populaires, a évolué progressivement vers le Lisp. Les idées 1-5 sont maintenant répandues. Le numéro 6 commence à apparaître dans le courant dominant. Python a une forme de 7, bien qu'il ne semble pas y avoir de syntaxe pour cela.

En ce qui concerne le numéro 8, c'est peut-être le plus intéressant du lot. Les idées 8 et 9 ne sont devenues une partie de Lisp que par accident, parce que Steve Russell a mis en œuvre quelque chose que McCarthy n'avait jamais eu l'intention de mettre en œuvre. Et pourtant, ces idées s'en sont révélées responsables à la fois de l'apparence étrange de Lisp et de ses caractéristiques les plus distinctives. Lisp a l'air étrange non pas tant parce qu'il a une syntaxe étrange que parce qu'il n'a pas de syntaxe ; vous exprimez des programmes directement dans les arbres d'analyse qui sont construits dans les coulisses lorsque d'autres langages sont analysés, et ces arbres sont faits de listes, qui sont des structures de données Lisp.

Exprimer le langage dans ses propres structures de données s'avère être une caractéristique très puissante. Les idées 8 et 9 ensemble signifient que vous pouvez écrire des programmes qui écrivent des programmes. Cela peut sembler une idée bizarre, mais c'est une chose quotidienne dans Lisp. La façon la plus courante de le faire est d'appeler une *macro*.

Le terme « macro » ne signifie pas en Lisp ce qu'il signifie dans d'autres langages. Une macro Lisp peut être n'importe quoi, d'une abréviation à un compilateur pour un nouveau langage. Si vous voulez vraiment comprendre Lisp, ou tout simplement élargir vos horizons de programmation, j'en apprendrais plus sur les macros.

Les macros (au sens de Lisp) sont toujours, pour autant que je sache, uniques à Lisp. C'est en partie parce que pour avoir des macros, vous devez probablement rendre votre langage aussi étrange que Lisp. C'est peut-être aussi parce que si vous ajoutez cette dernière augmentation de puissance, vous ne pouvez plus prétendre avoir inventé un nouveau langage, mais seulement un nouveau dialecte de Lisp.

Je mentionne cela principalement comme une blague, mais c'est tout à fait vrai. Si vous définissez un langage qui a car, cdr, cons, quote, cond, atom, eq, et une notation pour les fonctions exprimées sous forme de listes, alors vous pouvez en construire tout le reste de Lisp. C'est en fait la qualité déterminante de Lisp : c'était pour faire en sorte que McCarthy donne à Lisp la forme qu'il a.

Où les langages comptent

Même si Lisp représente une sorte de limite que les langages grand public approchent de manière asymptotique, cela signifie-t-il que vous devriez réellement l'utiliser pour écrire des logiciels ? Combien perdez-vous en utilisant un langage moins puissant ? N'est-il pas plus sage, parfois, de ne pas être à la limite de l'innovation ? Et la popularité d'une certaine tente n'est-elle pas sa propre justification ? Le patron aux cheveux pointus n'a-t-il pas raison, par exemple, de vouloir utiliser un langage pour lequel il peut facilement embaucher des programmeurs ?

Il y a, bien sûr, des projets où le choix du langage du programme n'a pas beaucoup d'importance. En règle générale, plus l'application est exigeante, plus vous obtenez d'effet de levier en utilisant un langage puissant. Mais beaucoup de projets ne sont pas du tout exigeants. La plupart des programmes consistent probablement à écrire de petits programmes de colle, et pour les petits programmes de colle, vous pouvez utiliser n'importe quel langage que vous connaissez déjà et qui a de bonnes bibliothèques pour tout ce que vous devez faire. Si vous avez juste besoin d'alimenter des données d'une application Windows à une autre, bien sûr, utilisez Visual Basic.

Vous pouvez aussi écrire de petits programmes de colle dans Lisp (je l'utilise comme calculatrice de bureau), mais la plus grande victoire pour des langages comme Lisp est à l'autre extrémité du spectre, où vous devez écrire des programmes sophistiqués pour résoudre des problèmes difficiles face à une concurrence féroce. Un bon exemple est le programme de recherche de tarifs aériens que ITA Software concède sous licence à Orbitz. Ces gars-là sont entrés sur un marché déjà dominé par deux grands concurrents enracinés, Travelocity et Expedia, et semblent les avoir humiliés sur le plan technologique.

Le cœur de l'application de l'ITA est un programme Common Lisp de 200 000 lignes qui recherche de nombreux ordres de grandeur plus de possibilités que leurs concurrents, qui utilisent apparemment encore des techniques de programmation de l'ère principale du cadre. Je n'ai jamais vu le code de l'ITA, mais selon l'un de leurs meilleurs pirates informatiques, ils utilisent beaucoup de macros, et je ne suis pas surpris de l'entendre.

Forces centripètes

Je ne dis pas qu'il n'y a aucun coût à utiliser des technologies inhabituelles. Le patron aux cheveux pointus ne se trompe pas complètement en s'inquiétant à ce sujet. Mais parce qu'il ne comprend pas les risques, il a tendance à les amplifier.

Je peux penser à trois problèmes qui pourraient découler de l'utilisation de langages moins courants. Vos programmes peuvent ne pas bien fonctionner avec des programmes écrits dans d'autres langages. Vous avez peut-être moins de bibliothèques à votre disposition. Et vous pourriez avoir du mal à embaucher des programmeurs.

Quelle est l'ampleur du problème de chacun d'entre eux ? L'importance du premier varie selon que vous avez le contrôle sur l'ensemble du système. Si vous écrivez un logiciel qui doit s'exécuter sur la machine d'un utilisateur distant sur un système d'exploitation propriétaire bogué (je ne mentionne aucun nom), il peut y avoir des avantages à écrire votre application dans le même langage que le système d'exploitation. Mais si vous contrôlez l'ensemble du système et que vous avez le code source de toutes les parties, comme le fait probablement ITA, vous pouvez utiliser les langages que vous voulez. En cas d'incompatibilité, vous pouvez la corriger vous-même.

Dans les applications basées sur serveur, vous pouvez vous en tirer en utilisant les technologies les plus avancées, et je pense que c'est la principale cause de ce que Jonathan Erickson appelle la "renaissance du langage de programmation". C'est pourquoi nous entendons même parler de nouveaux langages comme Perl et Python. Nous n'entendons pas parler de ces langages - parce que les gens les utilisent pour écrire des applications Windows, mais parce que les gens les utilisent sur des serveurs. Et à mesure que les logiciels passeront du bureau aux serveurs (un avenir à quoi même Microsoft semble résigné), il y aura de moins en moins de pression pour utiliser des technologies de milieu de route.

En ce qui concerne les bibliothèques, leur importance dépend également de l'application. Pour les problèmes moins exigeants, la disponibilité des bibliothèques peut l'emporter sur la puissance intrinsèque du langage. Où est le

seuil de rentabilité ? Difficile à dire exactement, mais où qu'il soit, il manque tout ce que vous seriez susceptible d'appeler une demande. Si une entreprise se considère comme une entreprise dans le secteur des logiciels, et qu'elle écrit une application qui sera l'un de ses produits, alors cela impliquera probablement plusieurs hackers et prendra au moins six mois pour écrire. Dans un projet de cette taille, les langages puissants commencent probablement à l'emporter sur la commodité des bibliothèques préexistantes.

La troisième inquiétude du patron aux cheveux pointus est la difficulté d'embaucher les programmeurs, je pense qu'il s'agit d'un faux-fuyant. Après tout, combien de hackers devez-vous embaucher ? Nous savons sûrement maintenant que les produits de logiciels sont mieux développés par des équipes de moins de dix personnes. Et vous ne devriez pas avoir de mal à embaucher des hackers à cette échelle pour n'importe quel langage dont quelqu'un a jamais entendu parler. Si vous ne trouvez pas dix hackers Lisp, alors votre entreprise est probablement basée dans la mauvaise ville pour développer des logiciels.

En fait, le choix d'un langage plus puissant diminue probablement la taille de l'équipe dont vous avez besoin, car (a) si vous utilisez un langage plus puissant, vous n'aurez probablement pas besoin d'autant de pirates, et (b) les pirates qui travaillent dans des langages plus avancés sont susceptibles d'être plus intelligents.

Je ne dis pas que vous n'aurez pas beaucoup de pression pour utiliser ce qui est perçu comme des technologies « standard ». Chez Viaweb, nous avons soulevé des sourcils auprès des investisseurs de capital-risque et des acquéreurs potentiels en utilisant Lisp. Mais nous avons également soulevé des sourcils en utilisant des boîtes Intel génériques comme serveurs au lieu de serveurs de "force industrielle" comme Suns, pour avoir utilisé un Unix open source alors obscur appelé FreeBSD au lieu d'un véritable système d'exploitation commercial comme Windows NT, pour avoir ignoré un supposé e-commerce standard appelé SET dont personne ne se souvient maintenant, et ainsi de suite.

Vous ne pouvez pas laisser les costumes prendre des décisions techniques pour vous. A-t-il alarmé les acquéreurs potentiels que nous ayons utilisé Lisp ? Certains, légèrement, mais si nous n'avions pas utilisé Lisp, nous n'aurions pas

été en mesure d'écrire le logiciel qui leur a donné envie de nous acheter. Ce qui leur semblait être une anomalie était en fait la cause et l'effet.

Si vous démarrez une start-up, ne concevez pas votre produit pour plaire aux investisseurs de capital-risque ou aux acquéreurs potentiels. Concevez votre produit pour plaire aux utilisateurs. Si vous gagnez les utilisateurs, tout le reste suivra. Et si vous ne le faites pas, personne ne se souciera de l'orthodoxie de vos choix technologiques.

Le Coût d'être Moyen

Combien perdez-vous en utilisant un langage moins puissant ? Il y a en fait des données à ce sujet.

La mesure de puissance la plus pratique est probablement la taille du code. Le but des langages de haut niveau est de vous donner de plus grandes abstractions - des briques plus grandes, pour ainsi dire, de sorte que vous n'en avez pas besoin autant pour construire un mur d'une taille donnée. Ainsi, plus le langage est puissant, plus le programme est court (pas simplement en caractères, bien sûr, mais en éléments distincts).

Comment un langage plus puissant vous permet-il d'écrire des programmes courts ? Une technique que vous pouvez utiliser, si le langage le permet, est ce qu'on appelle la programmation ascendante. Au lieu de simplement écrire votre application dans le langage de base, vous construisez sur le langage de base un langage pour écrire des programmes comme le vôtre, puis vous y écrivez votre programme. Le code combiné peut être beaucoup plus court que si vous aviez écrit l'ensemble de votre programme dans le langage de base - en fait, c'est ainsi que fonctionnent la plupart des algorithmes de compression. Un programme ascendant devrait également être plus facile à modifier, car dans de nombreux cas, la couche de langue n'aura pas à changer du tout.

La taille du code est importante, car le temps nécessaire à l'écriture d'un programme dépend principalement de sa longueur. Si votre programme est trois fois plus long dans un autre langage, il faudra trois fois plus de temps pour écrire - et vous ne pouvez pas contourner cela en embauchant plus de personnes,

car au-delà d'une certaine taille, les nouvelles embauches sont en fait une perte nette. Fred Brooks a décrit ce phénomène dans son célèbre livre *The Mythical Man-Month*, et tout ce que j'ai vu a eu tendance à confirmer ce qu'il a dit.

Alors, combien vos programmes sont-ils plus courts si vous les écrivez en Lisp ? La plupart des chiffres que j'ai entendus pour Lisp contre C, par exemple, ont été autour de 7 à 10 fois. Mais un article récent sur l'ITA dans le magazine New Architect a déclaré qu'"une ligne de Lisp peut remplacer 20 lignes de C", et puisque cet article était plein de citations du président de l'ITA, je suppose qu'ils ont obtenu ce numéro de l'ITA [6] Si c'est le cas, alors nous pouvons lui accorder une certaine confiance ; le logiciel de l'ITA comprend beaucoup de C et de C++ ainsi que Lisp, ils en parlent donc en connaissance de cause.

Je suppose que ces multiples ne sont même pas constants. Je pense qu'ils augmentent lorsque vous faites face à des problèmes plus difficiles et aussi lorsque vous avez des programmeurs plus intelligents. Un très bon hacker peut en extraire plus de meilleurs outils.

En tant que point de données sur la courbe, en tout cas, si vous deviez rivaliser avec l'ITA et que vous choisissiez d'écrire votre logiciel en C, ils seraient en mesure de développer des logiciels vingt fois plus rapidement que vous. Si vous passiez un an sur une nouvelle fonctionnalité, ils seraient en mesure de la dupliquer en moins de trois semaines. Alors que s'ils ne passaient que trois mois à développer quelque chose de nouveau, il faudrait cinq ans avant que vous ne l'obteniez aussi.

Et vous savez quoi ? C'est le meilleur scénario. Lorsque vous parlez de ratios de taille de code, vous supposez implicitement que vous pouvez réellement écrire le programme dans le langage le plus faible. Mais en fait, il y a des limites à ce que les programmeurs peuvent faire. Si vous essayez de résoudre un problème difficile avec un langage de niveau trop bas, vous atteignez un point où il y a tout simplement trop de choses à garder dans votre tête à la fois.

Donc, quand je dis qu'il faudrait cinq ans au concurrent imaginaire de l'ITA pour dupliquer quelque chose que l'ITA pourrait écrire en Lisp en trois mois, je veux dire cinq ans si rien ne va mal. En fait, la façon dont les choses

fonctionnent dans la plupart des entreprises, tout projet de développement qui prendrait cinq ans ne sera probablement jamais terminé du tout.

J'admetts qu'il s'agit d'un cas extrême. Les hackers de l'ITA semblent généralement peu intelligents, et C est un langage de niveau assez bas. Mais dans un marché concurrentiel, même un différentiel de deux ou trois contre un serait suffisant pour garantir que vous seriez toujours en retard.

Une recette

C'est le genre de possibilité à laquelle le patron aux cheveux pointus ne veut même pas penser. Et donc la plupart d'entre eux ne le font pas. Parce que, vous savez, quand il s'agit de ça, le patron aux cheveux pointus ne s'en soucie pas si son entreprise se fait botter le cul, tant que personne ne peut prouver que c'est de sa faute. Le plan le plus sûr pour lui personnellement est de rester près du centre d'élevage.

Au sein des grandes organisations, l'expression utilisée pour décrire cette approche est « les meilleures pratiques de l'industrie ». Son but est de protéger le patron aux cheveux pointus de la responsabilité : s'il choisit quelque chose qui est une "meilleure pratique de l'industrie" et que l'entreprise perd, il ne peut pas être blâmé. Il n'a pas choisi, l'industrie l'a fait.

Je crois que ce terme a été utilisé à l'origine pour décrire les méthodes comptables et ainsi de suite. Ce que cela signifie, en gros, c'est de ne rien faire de bizarre. Et en comptabilité, c'est probablement une bonne idée. Les termes « pointe » et « comptabilité » ne sonnent pas bien ensemble. Mais lorsque vous importez ce critère dans les décisions sur la technologie, vous commencez à obtenir de mauvaises réponses.

La technologie *devrait* souvent être de pointe. Dans les langages de programmation, comme Erann Gat l'a souligné, ce que les "meilleures pratiques de l'industrie" permettent d'obtenir n'est pas le meilleur, mais simplement la moyenne. Lorsqu'une décision vous amène à développer des logiciels à une fraction du taux de concurrents plus agressifs, les "meilleures pratiques" ne semblent pas vraiment être le bon nom pour cela.

Nous avons donc ici deux informations qui, je pense, sont très précieuses. En fait, je le sais de ma propre expérience. Numéro 1, les langages varient en puissance. Numéro 2, la plupart des gestionnaires l'ignorent délibérément. Entre eux, ces deux faits sont littéralement une recette pour gagner de l'argent. ITA est un exemple de cette recette en action. Si vous voulez gagner dans une entreprise de logiciels, il vous suffit de vous en prendre au problème le plus difficile que vous puissiez trouver, d'utiliser le langage le plus puissant que vous puissiez obtenir et d'attendre que les patrons aux cheveux pointus de vos concurrents reviennent à la moyenne.

Annexe : Puissance

Pour illustrer ce que je veux dire à propos du pouvoir relatif de la programmation des langues, considérez le problème suivant. Nous voulons écrire une fonction qui génère des accumulateurs - une fonction qui prend un nombre n, et renvoie une fonction qui prend un autre nombre i et renvoie n incrémenté par i. (C'est incrémenté de, pas plus. Un accumulateur doit s'accumuler.)

Dans Common Lisp [7], ce serait :

```
(defun foo (n)
  (lambda (i) (incf n i)))
```

En Ruby, c'est presque identique :

```
def foo (n)
  lambda {|i| n += i } end
```

Alors que dans Perl 5, c'est

```
sub foo {
  my ($n) = @_;
  sub {$n += shift}
}
```

qui a plus d'éléments que la version Lisp/Ruby parce que vous devez extraire les paramètres manuellement dans Perl.

Dans Smalltalk, le code est également légèrement plus long que dans Lisp et Ruby :

```
foo: n
| s |
s := n.
^[:i| s := s+i. ]
```

Parce que bien que les variables lexicales générales fonctionnent, vous ne pouvez pas faire une affectation à un paramètre, vous devez donc créer une nouvelle variable s pour contenir la valeur accumulée.

En Javascript, l'exemple est, encore une fois, légèrement plus long, car Javascript conserve la distinction entre les instructions et les expressions, vous avez donc besoin d'instructions de retour explicites pour renvoyer des valeurs :

```
function foo(n) {
  return function (i) {
    return n += i } }
```

(Pour être juste, Perl conserve également cette distinction, mais la traite de manière typique de Perl en vous laissant omettre les retours.)

Si vous essayez de traduire le code Lisp/Ruby/Perl/Smalltalk/Javascript en Python, vous rencontrez certaines limitations. Parce que Python ne prend pas entièrement en charge les variables lexicales, vous devez créer une structure de données pour contenir la valeur de n. Et bien que Python ait un type de données de fonction, il n'y a pas de représentation littérale pour un (à moins que le corps ne soit qu'une seule expression), vous devez donc créer une fonction nommée pour revenir. C'est ce que vous vous retrouvez avec :

```
def foo(n):
    s = [n]
    def bar(i):
        s[0] += i
        return s[0]
    return bar
```

Les utilisateurs de Python pourraient légitimement se demander pourquoi ils ne peuvent pas simplement écrire

```
def foo(n):
    return lambda i: return n += i
```

Ou même

```
def foo(n):
    lambda i: n += i
```

Et je suppose qu'ils le feront probablement, un jour. (Mais s'ils ne veulent pas attendre que Python évolue le reste du chemin vers Lisp, ils pourraient toujours juste...)

Dans les langages OO, vous pouvez, dans une mesure limitée, simuler une fermeture (une fonction qui fait référence à des variables définies dans le code environnant) en définissant une classe avec une méthode et un champ pour remplacer chaque variable à partir d'une portée englobante. Cela fait que le programme fait le genre d'analyse de code qui serait fait par le compilateur dans un langage avec une prise en charge complète de la portée lexicale, et cela ne fonctionnera pas si plus d'une fonction fait référence à la même variable, mais c'est suffisant dans des cas simples comme celui-ci.

Les experts de Python semblent convenir que c'est le moyen préféré de résoudre le problème en Python, en écrivant soit :

```

def foo(n):
    class acc:
        def __init__(self, s):
            self.s = s
        def inc(self, i):
            self.s += i
            return self.s
    return acc(n).inc

```

Ou

```

class foo:
    def __init__(self, n):
        self.n = n
    def __call__(self, i):
        self.n += i
        return self.n

```

Je les inclus parce que je ne voudrais pas que les défenseurs de Python disent que je déformais le langage, mais les deux me semblent plus complexes que la première version. Vous faites la même chose, vous mettez en place un endroit séparé pour tenir l'accumulateur ; c'est juste un champ dans un objet au lieu de l'en-tête d'une liste. Et l'utilisation de ces noms de champs spéciaux et réservés, en particulier `__call__`, semble un peu un hack.

Dans la rivalité entre Perl et Python, l'affirmation des hackers Python semble être que Python est une alternative plus élégante à Perl, mais ce que ce cas montre, c'est que la puissance est l'élément ultime : le programme Perl est plus simple (a moins d'éléments), même si la syntaxe est un peu plus laide.

Que diriez-vous des autres langages ? Dans les autres langages mentionnés ici - Fortran, C, C++, Java et Visual Basic - il ne semble pas que vous puissiez résoudre ce problème du tout. Ken Anderson dit que c'est à peu près aussi proche que possible en Java :

```

public interface Inttoint {
    public int call(int i);
}

public static Inttoint foo(final int n) {
    return new Inttoint() {
        int s = n;
        public int call(int i) {
            s = s + i;
            return s;
        }
    };
}

```

Ce qui n'est pas en deçà de la spécification parce qu'il ne fonctionne que pour les nombres entiers. Il n'est pas littéralement vrai que vous ne pouvez pas résoudre ce problème dans d'autres langages, bien sûr. Le fait que tous ces langages soient équivalents au Turing signifie que, à proprement parler, vous pouvez écrire n'importe quel programme dans n'importe lequel d'entre eux. Alors, comment le feriez-vous ? Dans le cas limite, en écrivant un interpréteur Lisp dans la langue la moins puissante.

Cela ressemble à une blague, mais cela arrive si souvent à des degrés divers dans les grands projets de programmation qu'il y a un nom pour ce phénomène, la Greenspun's Tenth Rule :

Tout programme C ou Fortran suffisamment compliqué contient une mise en œuvre lente ad hoc de la moitié de Common Lisp.

Si vous essayez de résoudre un problème difficile, la question n'est pas de savoir si vous utiliserez un langage assez puissant, mais si vous (a) utiliserez un langage puissant, (b) écrirez un interpréteur de facto pour un, ou (c) vous-même devenir un compilateur humain pour un. Nous voyons que cela commence déjà à se produire dans l'exemple Python, où nous simulons en fait le code qu'un compilateur générera pour implémenter une variable lexicale.

Cette pratique est non seulement courante, mais aussi institutionnalisée. Par exemple, dans le monde OO, vous entendez beaucoup parler de "modèles".

Je me demande si ces modèles ne sont pas parfois une preuve du cas (c), le compilateur humain, au travail [8] Quand je vois des modèles dans mes programmes, je considère que c'est un signe de problème. La forme d'un programme ne doit refléter que le problème qu'il doit résoudre. Toute autre régularité dans le code est un signe, du moins pour moi, que j'utilise des abstractions qui ne sont pas assez puissantes - souvent que je génère à la main les extensions d'une macro que je dois écrire.

Chapitre 14

Le Langage de Rêve

“De toutes les tyrannies, une tyrannie exercée pour le bien de ses victimes peut être la plus oppressive.”

C. S. Lewis

Un de mes amis a dit un jour à un éminent système d'exploitation expert qu'il voulait concevoir un très bon langage de programmation. L'expert a déclaré que ce serait une perte de temps, que les langages de programmation ne deviennent pas populaires ou impopulaires en fonction de leurs mérites, et donc peu importe à quel point son langage était bon, personne ne l'utilisera. Au moins, c'est ce qui était arrivé au langage qu'il avait conçu.

Qu'est-ce qui rend un langage populaire ? Les langages populaires méritent-ils leur popularité ? Vaut-il la peine d'essayer de définir un bon langage de programmation ? Comment le feriez-vous ?

Je pense que les réponses à ces questions peuvent être trouvées en regardant les hackers et en apprenant ce qu'ils veulent. Les langages de programmation sont pour les hackers, et un langage de programmation est bon pour programmer un langage (plutôt que, par exemple, un exercice de sémantique dénotationnelle ou de conception de compilateur) si et seulement si les hackers l'aiment.

La mécanique de la popularité

Il est vrai, certainement, que la plupart des gens ne choisissent pas les langages de programmation simplement en fonction de leurs mérites. La plupart des programmeurs se font dire quelle langue utiliser par quelqu'un d'autre. Et pourtant, je pense que l'effet de tels facteurs externes sur la popularité des langages de programmation n'est pas aussi grand qu'on le pense parfois. Je pense qu'un des plus gros problème est que l'idée d'un hacker d'un bon langage de programmation n'est pas la même que celle de la plupart des concepteurs de langage.

Entre les deux, l'opinion du hacker est celle qui compte. Les langages de programmation ne sont pas des théorèmes. Ce sont des outils, conçus pour les gens, et ils doivent être conçus pour s'adapter aux forces et aux faiblesses humaines autant que les chaussures doivent être conçues pour les pieds humains. Si une chaussure se pince lorsque vous la mettez, c'est une mauvaise chaussure, aussi élégante soit-elle comme un morceau de sculpture.

Il se peut que la majorité des programmeurs ne puissent pas distinguer un bon langage d'un mauvais langage. Mais ce n'est pas différent avec n'importe quel autre outil. Cela ne signifie pas que c'est une perte de temps d'essayer de concevoir un bon langage. Les hackers experts peuvent dire si c'est un bon langage quand ils en voient un, et ils l'utiliseront. Les hackers experts sont une infime minorité, certes, mais cette petite minorité écrit tous les bons logiciels, et leur influence est telle que le reste des programmeurs auront tendance à utiliser n'importe quel langage qu'ils utilisent. Souvent, en effet, ce n'est pas seulement de l'influence, mais du commandement : souvent, les hackers experts sont les personnes mêmes qui, en tant que patrons ou conseillers du corps professoral, disent aux autres programmeurs quel langage utiliser.

L'opinion des hackers experts n'est pas la seule force qui dissuade la popularité relative des langages de programmation - les logiciels hérités (Fortran, Cobol) et le battage médiatique (Ada, Java) jouent également un rôle - mais je pense que c'est la force la plus puissante à long terme. Compte tenu d'une masse critique initiale et de suffisamment de temps, un langage de programmation devient probablement aussi populaire qu'il le mérite. Et la clarté populaire sépare davantage les bons langages des mauvais, car les commentaires des vrais utilisateurs en direct conduisent toujours à des améliorations. Regardez à quel point un langage populaire a changé au cours de sa vie. Perl et Fortran sont des cas extrêmes, mais même Lisp a beaucoup changé.

Donc, est-ce qu'un langage doit être bon ou non pour être populaire ? Je pense qu'un langage doit être populaire pour être bon. Et il doit rester populaire pour rester bon. L'état de l'art des langages de programmation ne s'arrête pas. Bien qu'il y ait peu de changement dans les profondeurs de la mer, dans les caractéristiques du langage de base, il y en a beaucoup à la surface, dans des choses comme les bibliothèques et les environnements.

Bien sûr, les hackers doivent connaître un langage avant de pouvoir l'utiliser. Comment vont-ils entendre ? D'autres pirates informatiques. Mais il doit y avoir un groupe initial de hackers qui utilisent le langage pour que d'autres en entendent parler. Je me demande quelle doit être la taille de ce groupe ; combien d'utilisateurs font une masse critique ? Du haut de ma tête, je dirais vingt. Si un langage avait vingt utilisateurs distincts, c'est-à-dire vingt utilisateurs qui décidaient eux-mêmes de l'utiliser, je le considérerais comme réel.

S'y rendre ne peut pas être facile. Je ne serais pas surpris s'il est plus difficile de passer de zéro à vingt que de vingt à mille. La meilleure façon d'obtenir ces vingt premiers utilisateurs est probablement un cheval de Troie : donnez aux gens une application qu'ils veulent, qui se trouve être écrite dans le nouveau langage.

Facteurs externes

Commençons par reconnaître un facteur externe qui affecte la popularité d'un langage de programmation. Pour devenir populaire, un langage de programmation doit être le langage de script d'un système populaire. Fortran et Cobol étaient les langages de script des premiers mainframes IBM. C'était le langage de script d'Unix, et donc, plus tard, Perl et Python. Tcl est le langage de script de Tk, Visual Basic de Windows, (une forme de) Lisp d'Emacs, PHP des serveurs web, et Java et Javascript des navigateurs web.

Les langages de programmation n'existent pas isolément. Hacker est un verbe transitif - les hackers bidouillent généralement quelque chose - et dans la pratique, les langages sont jugés par rapport à tout ce qu'ils ont l'habitude de bidouiller. Donc, si vous voulez concevoir un langage populaire, vous devez soit fournir plus qu'un langage, soit concevoir votre langage pour remplacer le langage de script d'un système existant.

Une façon de décrire cette situation est de dire qu'un langage n'est pas jugé sur ses propres mérites. Un autre point de vue est qu'un langage de programmation n'est vraiment pas un langage de programmation à moins que ce ne soit aussi le langage de script de quelque chose. Cela ne semble injuste que si cela se présente comme une surprise. Je pense que ce n'est pas plus injuste que

de s'attendre à un langage de programmation pour avoir, par exemple, une implémentation. Ce n'est qu'une partie de ce qu'est un langage de programmation.

Un langage de programmation a besoin d'une bonne implémentation, bien sûr, et cela doit être gratuit. Les entreprises paieront pour les logiciels, mais les hackers individuels ne le feront pas, et ce sont les hackers que vous devez attirer.

Un langage doit également avoir un livre à ce sujet. Le livre doit être mince, bien écrit et plein de bons exemples. Le *langage de programmation C* de Kernighan et Ritchie est l'idéal ici. Pour le moment, je dirais presque qu'un langage doit avoir un livre publié par O'Reilly. Cela devient le test de l'importance pour les hackers.

Il devrait également y avoir de la documentation en ligne. En fait, le livre peut commencer par une documentation en ligne. Mais les livres physiques ne sont pas encore obsolètes. Leur format est pratique, et la censure de fait imposée par les éditeurs est un filtre utile, même s'il est imparfait. Les librairies sont l'un des endroits les plus importants pour apprendre de nouveaux langages.

Concision

Étant donné que vous pouvez fournir les trois choses dont n'importe quel langage a besoin - une implémentation gratuite, un livre et quelque chose à pirater - comment faites-vous un langage que les hackers aimeront ?

Une chose que les hackers aiment, c'est la concision. Les hackers sont paresseux, de la même manière que les mathématiciens et les architectes modernistes sont paresseux : ils détestent tout ce qui est étranger. Ce ne serait pas loin de la vérité de dire qu'un pirate sur le point d'écrire un programme décide quel langage utiliser, au moins inconsciemment, en fonction du nombre total de caractères qu'il devra taper. Si ce n'est pas exactement ce que pensent les hackers, un concepteur de langage ferait bien d'agir comme si c'était le cas.

Le type le plus important de succinctance vient de rendre la langue plus abstraite. C'est pour obtenir cela que nous utilisons des langages de haut niveau

en premier lieu. Il semblerait donc que plus vous en obtenez, mieux c'est. Un concepteur de langage devrait toujours regarder les programmes et se demander, y a-t-il un moyen d'exprimer cela en moins de jetons ? Si vous pouvez faire quelque chose qui rend différents programmes plus courts, ce n'est probablement pas une coïncidence : vous avez probablement découvert une nouvelle abstraction utile.

C'est une erreur d'essayer d'enfanter l'utilisateur avec des expressions longues destinées à ressembler à l'anglais. Cobol est connu pour cette faille. Un hacker envisagerait qu'on lui demande d'écrire...

ajouter x à y en donnant z

Au lieu de :

$$Z = X + Y$$

... comme quelque chose entre une insulte à son intelligence et un péché contre Dieu.

La concision est un endroit où les langages dactylographiés statiquement perdent. Toutes choses étant égales par ailleurs, personne ne veut commencer un programme avec un tas de déclarations. Tout ce qui peut être implicite, devrait l'être. La quantité de boilerplate dans un programme Java hello-world est presque suffisante pour condamner [1].

Les jetons individuels doivent également être courts. Perl et Common Lisp occupent des pôles opposés sur cette question. Les programmes Perl peuvent être cryptiquement denses, tandis que les noms des opérateurs Common Lisp intégrés sont comiquement longs. Les concepteurs de Common Lisp s'attendaient probablement à ce que les utilisateurs aient des éditeurs de texte qui taperaient ces longs noms pour eux. Mais le coût d'un nom long n'est pas seulement le coût de sa saisie. Il y a aussi le coût de sa lecture et le coût de l'espace qu'il occupe sur votre écran.

Hackability

Il y a une chose plus importante que la concision pour un hacker : être capable de faire ce que vous voulez. Dans l'histoire des langages de programmation, un effort surprenant a été consacré à empêcher les programmeurs de faire des choses considérées comme inappropriées. C'est un plan dangereusement présomptueux. Comment le concepteur de langage peut-il savoir ce que le programmeur devra faire ? Je pense que les concepteurs de langages feraient mieux de considérer leur utilisateur cible comme un génie qui aura besoin de faire des choses qu'ils n'avaient jamais prévues, plutôt qu'un bouffon qui a besoin d'être protégé de lui-même. Le bumbler se tirerait dans le pied de toute façon. Vous pouvez l'éviter de se référer à des variables dans un autre module, mais vous ne pouvez pas l'éviter d'écrire un programme mal conçu pour résoudre le mauvais problème, et de prendre une éternité pour le faire.

Les bons programmeurs veulent souvent faire des choses dangereuses et peu recommandables. Par désagréable, je veux dire des choses qui vont derrière n'importe quelle façade sémantique que le langage essaie de présenter : s'accrocher à la représentation interne d'une certaine abstraction de haut niveau, par exemple. Les hackers aiment hacker, et le bidouillage signifie entrer dans les choses et deviner le concepteur original.

Laissez-vous deviner. Lorsque vous fabriquez un outil, les gens l'utilisent d'une manière que vous n'aviez pas l'intention de faire, et c'est particulièrement vrai pour un outil très articulé comme un langage de programmation. Beaucoup de hackers voudront modifier votre modèle sémantique d'une manière que vous n'auriez jamais imaginée. Je dis, laissez-les. Donnez au programmeur l'accès à autant de choses internes que possible.

Un hacker ne peut vouloir subvertir le modèle prévu des choses qu'une ou deux fois dans un grand programme. Mais quelle différence cela fait d'être capable de le faire. Et il peut s'agir peut-être plus qu'une question de résolution d'un problème. Il y a une sorte de plaisir ici aussi. Les hackers partagent le plaisir secret du chirurgien à fouiller dans les entrailles grossières, le plaisir secret de l'adolescent à faire éclater des boutons [2] Pour les garçons, au moins, ce genre d'horreurs est fascinant. Le magazine *Maxim* publie un volume annuel

de photographies, contenant un mélange de pin-ups et d'accidents macabres. Ils connaissent leur public.

Un très bon langage devrait être à la fois propre et sale : proprement conçu, avec un petit noyau d'opérateurs bien compris et hautement orthogonal, mais sale dans le sens qu'il permet aux hackers de s'y frayer. C'est comme ça. Il en était de plus que les premiers Lisps. Le langage d'un vrai hacker aura toujours un caractère légèrement rafistolé.

Un bon langage de programmation devrait avoir des fonctionnalités qui font que le genre de personnes qui utilisent l'expression "ingénierie logicielle" secouent la tête avec désapprobation. À l'autre extrémité du continuum se trouvent des langages comme Pascal, des modèles de bienséance qui sont bons pour l'enseignement et pas grand-chose d'autre.

Programmes jetables

Pour être attrayant pour les hackers, un langage doit être bon pour écrire les types de programmes qu'ils veulent écrire. Et cela signifie, de manière surprenante, que cela doit être bon pour écrire des programmes jetables.

Un programme jetable est un programme que vous écrivez rapidement pour une tâche limitée : un programme pour automatiser une tâche d'administration du système, ou pour générer des données de test pour une simulation, ou pour convertir des données d'un format à un autre. Ce qui est surprenant avec les programmes jetables, c'est que, comme les bâtiments « temporaires » construits dans tant d'universités américaines pendant la Seconde Guerre mondiale, ils ne sont souvent pas jetés. Beaucoup évoluent vers de vrais programmes, avec de vraies fonctionnalités et de vrais utilisateurs.

J'ai l'impression que les meilleurs grands programmes commencent la vie de cette façon, plutôt que d'être conçus en grand dès le début, comme le barrage Hoover. C'est terrifiant de construire quelque chose de grand à partir de zéro. Lorsque les gens entreprennent un projet trop important, ils deviennent vite dépassés. Soit le projet s'enlise, soit le résultat est stérile et en bois : un centre commercial plutôt qu'un vrai centre-ville, Brasilia plutôt que Rome, Ada plutôt que C.

Une autre façon d'obtenir un grand programme est de commencer par un programme jetable et de continuer à l'améliorer. Cette approche est moins intimidante, et la conception du programme bénéficie de l'évolution. Les programmes qui ont évolué de cette façon sont probablement encore écrits dans n'importe quelle langue dans laquelle ils ont été écrits pour la première fois, car il est rare qu'un programme soit porté, sauf pour des raisons politiques. Et donc, paradoxalement, si vous voulez faire un langage qui est utilisé pour les grands systèmes, vous devez le rendre bon pour écrire des programmes jetables, parce que c'est de là que viennent les grands systèmes.

Perl est un exemple frappant de cette idée. Il n'était pas seulement conçu pour écrire des programmes jetables, mais c'était à peu près un programme jetable lui-même. Perl a commencé sa vie comme une collection d'utilitaires pour la génération de rapports, et n'a évolué pour devenir un langage de programmation qu'au fur et à mesure que les programmes jetables que les gens y ont écrits s'agrandissaient. Ce n'était pas avant Perl 5 (si alors) que le langage était approprié pour écrire des programmes sérieux, et pourtant il était déjà très populaire.

Qu'est-ce qui rend un langage bon pour les programmes jetables ? Pour commencer, il doit être facilement disponible. Un programme jetable est quelque chose que vous vous attendez à écrire dans une heure. Le langage doit donc probablement déjà être installé sur l'ordinateur que vous utilisez. Ce ne peut pas être quelque chose que vous devez installer avant de l'utiliser. Il doit être là. C'était là parce qu'il était venu avec le système d'exploitation. Perl était là parce qu'il s'agissait à l'origine d'un outil pour les administrateurs système, et le vôtre l'avait déjà installé.

Être disponible signifie plus que d'être installé, cependant. Un langage interactif, avec une interface en ligne de commande, est plus disponible qu'un langage que vous devez compiler et exécuter séparément. Un langage de programmation populaire doit être interactif et démarrer rapidement.

Une autre chose que vous voulez dans un programme jetable est la concision. C'est toujours attrayant pour les hackers, et jamais plus que dans un programme qu'ils s'attendent à sortir en une heure.

Bibliothèques

Bien sûr, le summum de la concision est d'avoir le programme prêt à l'emploi écrit pour vous, et simplement de l'appeler. Et cela nous amène à ce qui, je pense, sera une caractéristique de plus en plus importante de la programmation des langues : les bibliothèques. Perl gagne parce qu'il a de grandes bibliothèques pour manipuler les cordes. Cette classe de fonction de bibliothèque est particulièrement importante pour les programmes jetables, qui sont souvent écrits à l'origine pour convertir ou extraire des données. De nombreux programmes Perl commencent probablement par quelques appels de bibliothèque collés.

Je pense que beaucoup des progrès qui se produiront dans les langages de programmation au cours des cinquante prochaines années auront à voir avec les fonctions de bibliothèque. Je pense que les futurs langages de programmation auront des bibliothèques aussi soigneusement conçues que le langage de base. La conception du langage de programmation ne concernera pas la question de savoir s'il faut rendre votre langage typé statiquement ou dynamiquement, ou orienté objet, ou fonctionnel, ou quoi que ce soit, autant que sur la façon de concevoir de grandes bibliothèques. Le genre de concepteurs de langage qui aiment réfléchir à la façon de concevoir des systèmes de type peut frissonner à ce sujet. C'est presque comme écrire des applications ! Eh bien, tant pis. Les langages sont pour les programmeurs, et les bibliothèques sont ce dont les programmeurs ont besoin.

Il est difficile de concevoir de bonnes bibliothèques. Il ne s'agit pas simplement d'écrire beaucoup de code. Une fois que les bibliothèques sont trop grandes, il peut parfois prendre plus de temps pour trouver la fonction dont vous avez besoin que de l'écrire vous-même. Les bibliothèques doivent être conçues à l'aide d'un petit ensemble d'opérateurs orthogonaux, tout comme le langage de base. Il devrait être possible pour le programmeur de deviner quel appel de bibliothèque fera ce dont il a besoin.

Efficacité

Un bon langage, comme tout le monde le sait, devrait générer un code rapide. Mais en pratique, je ne pense pas que le code rapide provienne

principalement de choses que vous faites dans la conception du langage. Comme Knuth l'a souligné il y a longtemps, la vitesse n'a d'importance que dans certains goulets d'étranglement critiques. Et comme de nombreux programmeurs l'ont observé depuis, on se trompe souvent sur l'endroit où se trouvent ces goulets d'étranglement.

Donc, en pratique, la façon d'obtenir du code rapide est d'avoir un bon profileur, plutôt que, par exemple, de faire taper la langue statiquement. Vous n'avez pas besoin de connaître le type de chaque argument dans chaque appel du programme. Vous devez être en mesure de déclarer les types d'arguments dans les goulets d'étranglement. Et plus encore, vous devez être en mesure de savoir où se trouvent les goulets d'étranglement.

Une plainte que les gens ont eue avec des langages de très haut niveau comme Lisp est qu'il est difficile de dire ce qui est cher. C'est peut-être vrai. Cela pourrait également être inévitable, si vous voulez avoir un langage très abstrait. Et dans tous les cas, je pense qu'un bon profilage contribuerait grandement à résoudre le problème : vous apprendriez bientôt ce qui était cher.

Une partie du problème ici est sociale. Les concepteurs de langages aiment écrire des compilateurs rapides. C'est ainsi qu'ils mesurent leurs compétences. Ils considèrent le profileur comme un add-on, au mieux. Mais en pratique, un bon profileur peut faire plus pour améliorer la vitesse des programmes réels écrits dans le langage qu'un compilateur qui génère du code rapide. Ici, encore une fois, les concepteurs de langages sont quelque peu déconnectés de leurs utilisateurs. Ils font un très bon travail pour résoudre légèrement le mauvais problème.

Ce pourrait être une bonne idée d'avoir un profileur actif - pour pousser les données de formation au programmeur au lieu d'attendre qu'il le demande. Par exemple, l'éditeur pourrait afficher les goulets d'étranglement en rouge lorsque le programmeur modifie le code source. Une autre approche serait de représenter d'une manière ou d'une autre ce qui se passe dans la gestion des programmes. Ce serait une victoire particulièrement importante dans les applications basées sur serveur, où vous avez beaucoup de programmes en cours d'exécution à regarder. Un profileur actif pourrait montrer graphiquement ce qui

se passe dans la mémoire pendant l'exécution d'un programme, ou même émettre des sons qui indiquent ce qui se passe.

Le son est un bon indice pour les problèmes. Chez Viaweb, nous avions un grand tableau de cadrons montrant ce qui arrivait à nos serveurs web. Les mains étaient déplacées par de petits servomoteurs qui faisaient un léger bruit lorsqu'ils tournaient. Je ne pouvais pas voir le tableau depuis mon bureau, mais j'ai constaté que je pouvais dire immédiatement, par le son, quand il y avait un problème avec un serveur.

Il pourrait même être possible d'écrire un profileur qui détecterait automatiquement les algorithmes inefficaces. Je ne serais pas surpris si certains schémas d'accès à la mémoire s'avéraient être des signes sûrs de mauvais algorithmes. S'il y avait un petit gars qui courrait à l'intérieur de l'ordinateur pour exécuter nos programmes, il aurait probablement une histoire aussi longue et plaintive à raconter sur son travail en tant qu'employé du gouvernement fédéral. J'ai souvent l'impression d'envoyer le processeur sur beaucoup de poursuites d'oies sauvages, mais je n'ai jamais eu un bon moyen de regarder ce qu'il fait.

Un certain nombre de langages se compilent maintenant en code d'octets, qui est ensuite exécuté par un interpréteur. Ceci est généralement fait pour faciliter le portage de l'implémentation, mais cela pourrait être une fonctionnalité de langage utile. Ce pourrait être une bonne idée de faire du code octet une partie officielle du langage et de permettre aux programmeurs d'utiliser l'octet en ligne code dans les goulots d'étranglement. Alors de telles optimisations seraient également portables.

La nature de la vitesse, perçue par l'utilisateur final, peut changer. Avec l'essor des applications basées sur serveur, de plus en plus de programmes peuvent s'être liés aux Entrées/Sorties. Cela vaudra la peine de faire des E/S rapidement. Le langage peut aider avec des tâches simples comme des fonctions de sortie simples, rapides et formatées, ainsi qu'avec des changements structurels profonds comme la mise en cache et les objets persistants.

Les utilisateurs sont intéressés par le temps de réponse. Mais un autre type d'efficacité sera de plus en plus important : le nombre d'utilisateurs simultanés que vous pouvez prendre en charge par processeur. Bon nombre des

applications d'intérêt écrites à l'avenir seront basées sur un serveur, et le nombre d'utilisateurs par serveur est la question critique pour quiconque héberge de telles applications. Dans le coût en capital d'une entreprise offrant une application sur serveur, il s'agit du diviseur.

Pendant des années, l'efficacité n'a pas beaucoup d'importance dans la plupart des applications des utilisateurs finaux. Les développeurs ont pu supposer que les utilisateurs auraient des processeurs de plus en plus rapides assis sur leur bureau. Et la loi de Parkinson s'est avérée aussi puissante que la loi de Moore. Les logiciels se sont gonflés pour consommer les ressources disponibles. Cela changera avec les applications basées sur le serveur, car le matériel et les logiciels seront fournis ensemble. Pour les entreprises qui proposent des applications basées sur un serveur, le nombre d'utilisateurs qu'elles peuvent prendre en charge par serveur fera une grande différence.

Dans certaines applications, le processeur sera le facteur limitant, et la vitesse d'exécution sera la chose la plus importante à optimiser. Mais souvent, la mémoire sera la limite ; le nombre d'utilisateurs simultanés sera déterminé par la quantité de mémoire dont vous avez besoin pour les données de chaque utilisateur. La langue peut aussi aider ici. Une bonne prise en charge des threads permettra à tous les utilisateurs de partager un seul tas. Il peut également être utile d'avoir des objets persistants et/ou une prise en charge au niveau du langage pour le chargement paresseux.

Temps

Le dernier ingrédient dont un langage populaire a besoin est le temps. Personne ne veut écrire des programmes dans un langage qui pourrait disparaître, comme le font tant de langages de programmation. Ainsi, la plupart des hackers auront tendance à attendre qu'un langage existe depuis quelques années avant même de l'envisager.

Les inventeurs de nouvelles choses merveilleuses sont souvent surpris de couvrir cela, mais vous avez besoin de temps pour faire passer n'importe quel message aux gens. Un de mes amis fait rarement quoi que ce soit la première fois que quelqu'un lui demande. Il sait que les gens demandent parfois des choses qu'ils ne veulent pas. Pour éviter de perdre son temps, il attend la

troisième ou la quatrième fois qu'on lui demande de faire quelque chose. D'ici là, celui qui lui demande peut être assez ennuyé, mais au moins, ils veulent probablement vraiment tout ce qu'ils demandent.

La plupart des gens ont appris à faire une sorte de filtrage similaire sur les nouvelles choses dont ils entendent parler. Ils ne commencent même pas à faire attention tant qu'ils n'ont pas entendu parler de quelque chose dix fois. Ils sont parfaitement justifiés : la majorité des nouveaux projets en vogue s'avèrent être une perte de temps et finissent par disparaître. En retardant l'apprentissage du VRML, j'ai évité d'avoir à l'apprendre du tout.

Donc, toute personne qui invente quelque chose de nouveau doit s'attendre à répéter son message pendant des années avant que les gens ne commencent à le recevoir. Il nous a fallu des années pour faire passer aux gens que le logiciel de Viaweb n'avait pas besoin d'être téléchargé. La bonne nouvelle, c'est qu'une simple répétition résout le problème. Tout ce que vous avez à faire est de continuer à raconter votre histoire, et finalement les gens commenceront à l'entendre. Ce n'est pas quand les gens remarquent que vous êtes là qu'ils font attention ; c'est quand ils remarquent que vous êtes toujours là.

C'est tout aussi bien qu'il faut généralement un certain temps pour gagner du temps. La plupart des technologies évoluent beaucoup même après leur premier lancement, en particulier les langages de programmation. Rien ne pourrait être mieux pour une nouvelle technologie que quelques années d'utilisation seulement par un petit nombre de premiers utilisateurs. Les premiers utilisateurs sont sophistiqués et exigeants, et effacent rapidement tous les défauts qui subsistent dans votre technologie. Lorsque vous n'avez que quelques utilisateurs, vous pouvez être en contact étroit avec chacun d'entre eux. Et les premiers utilisateurs pardonnent lorsque vous améliorez votre système, même si cela provoque une certaine casse.

Il y a deux façons d'introduire une nouvelle technologie : la méthode de croissance organique et la méthode du big bang. La méthode de croissance organique est illustrée par la start-up de garage classique non financée par le siège du pantalon. Quelques gars, travaillant dans l'obscurité, développent de nouvelles technologies. Ils le lancent sans marketing et n'ont initialement que quelques utilisateurs (fanatiquement dévoués). Ils ont pour but d'améliorer la

technologie, et pendant ce temps, leur base d'utilisateurs grandit par le bouche à oreille. Avant qu'ils ne s'en rendent compte, ils sont grands.

L'autre approche, la méthode du big bang, est illustrée par la start-up fortement commercialisée soutenue par le capital-risque. Ils se précipitent pour développer un produit, le lancer avec beaucoup de publicité, et immédiatement (ils ont espoir) avoir une grande base d'utilisateurs.

En général, les gars du garage envient les gars du big bang. Les grand big bang guys sont lisses et confiants et respectés par les VCs. Ils peuvent se permettre le meilleur de tout, et la campagne de relations publiques qui arrondit le lancement a pour effet secondaire de les rendre célèbres. Les gars de la croissance organique, assis dans leur garage, se sentent pauvres et mal aimés. Et pourtant, je pense qu'ils se trompent souvent de se sentir désolés pour eux-mêmes. La croissance organique semble produire une meilleure technologie et des fondateurs plus riches que la méthode du big bang. Si vous regardez les technologies dominantes aujourd'hui, vous constaterez que la plupart d'entre elles ont grandi de manière organique.

Ce modèle ne s'applique pas seulement aux entreprises. Vous le voyez aussi dans la recherche. Multics et Ada étaient des projets big-bang, et Unix et C étaient des projets de croissance organique.

Transformer

« La meilleure écriture est la réécriture », a écrit E. B. Blanc. Tous les bons écrivains le savent, et c'est vrai aussi pour les logiciels. La partie la plus importante du design est la refonte. Les langages de programmation, en particulier, ne sont pas assez repensés.

Pour écrire de bons logiciels, vous devez garder simultanément deux idées d'opinion dans votre tête. Vous avez besoin de la foi naïve du jeune hacker en ses capacités, et en même temps le scepticisme du vétéran. Vous devez être capable de penser à quel point cela peut être difficile ? Avec une moitié de votre cerveau tout en pensant que cela ne fonctionnera jamais avec l'autre.

L'astuce est de réaliser qu'il n'y a pas de véritable contradiction ici. Vous voulez être optimiste et sceptique à propos de deux choses différentes. Vous devez être optimiste quant à la possibilité de résoudre le problème, mais sceptique quant à la valeur de la solution que vous avez jusqu'à présent.

Les gens qui font du bon travail pensent souvent que tout ce sur quoi ils travaillent n'est pas bon. D'autres voient ce qu'ils ont fait et pensent que c'est merveilleux, mais le créateur ne voit que des défauts. Ce modèle n'est pas une coïncidence : l'inquiétude a rendu le travail bon.

Si vous pouvez garder l'espoir et l'inquiétude équilibrés, ils feront avancer un projet de la même manière que vos deux jambes font avancer un vélo. Dans la première phase du moteur d'innovation à deux cycles, vous travaillez furieusement sur un problème, inspiré par votre confiance que vous serez en mesure de le résoudre. Dans la deuxième phase, vous regardez ce que vous avez fait à la lumière froide du matin et vous voyez très clairement tous ses défauts. Mais tant que votre esprit critique ne l'emporte pas sur votre espoir, vous serez en mesure de regarder votre système certes incomplet et de penser, à quel point peut-il être difficile d'obtenir le reste du chemin ?

Il est difficile de garder les deux forces équilibrées. Chez les jeunes hackers, l'optimisme prédomine. Ils produisent quelque chose, sont convaincus que c'est génial et ne l'améliorent jamais. Chez les vieux hackers, le scepticisme prédomine, et ils n'oseraient même pas se lancer dans des projets ambitieux.

Tout ce que vous pouvez faire pour maintenir le cycle de refonte est bon. La prose peut être réécrite encore et encore jusqu'à ce que vous en soyez satisfait. Mais les logiciels, en règle générale, ne sont pas assez redessinés. La prose a des lecteurs, mais les logiciels ont des utilisateurs. Si un écrivain réécrit un essai, il est peu probable que les personnes qui lisent la nouvelle version se plaignent que leurs pensées ont été brisées par une incompatible nouvellement introduite.

Les utilisateurs sont une épée à double tranchant. Ils peuvent vous aider à améliorer votre langage, mais ils peuvent aussi vous dissuader de l'améliorer. Choisissez donc vos utilisateurs avec soin et soyez lent à augmenter leur

nombre. Avoir des utilisateurs, c'est comme l'optimisation : le plus sage consiste à retarder l'échéance.

De plus, en règle générale, vous pouvez à tout moment vous en tirer en changeant plus que vous ne le pensez. Introduire le changement, c'est comme retirer un pansement : la douleur est un souvenir presque dès que vous la ressentez.

Tout le monde sait que ce n'est pas une bonne idée de faire signer un langage par un comité. Les comités produisent une mauvaise conception. Mais je pense que le pire danger des comités est qu'ils interfèrent avec la refonte. Il y a tellement de travail à introduire des changements que personne ne veut déranger. Tout ce qu'un comité décide a tendance à rester ainsi, même si la plupart des membres ne l'aiment pas.

Même un comité de deux personnes fait obstacle à la refonte. Cela se produit en particulier dans les interfaces entre les logiciels écrits par deux personnes différentes. Pour changer l'interface, les deux doivent accepter de la changer en même temps. Et donc, les interfaces ont tendance à ne pas changer du tout, ce qui est un problème parce qu'elles ont tendance à être l'une des parties les plus ad hoc de tout système.

Une solution ici pourrait être de concevoir des systèmes de sorte que les interfaces soient horizontales au lieu de verticales - de sorte que les modules soient toujours des strates d'abstraction statiquement empilées. Ensuite, l'interface aura tendance à appartenir à l'un d'eux. Le niveau inférieur de deux niveaux sera soit un langage dans lequel le niveau supérieur est écrit, auquel cas le niveau inférieur sera propriétaire de l'interface, soit un esclave, auquel cas l'interface peut être dictée par le niveau supérieur.

La langue des rêves

En résumé, essayons de décrire le langage de rêve du hacker. Le langage des rêves est propre et laconique. Il a un niveau supérieur interactif qui démarre rapidement [3]. Vous pouvez écrire des programmes pour résoudre des problèmes avec très peu de code. Presque tout le code de tout programme que

vous écrivez est un code spécifique à votre application. Tout le reste a été fait pour vous.

La syntaxe du langage est bref à un défaut. Vous n'avez jamais besoin de taper un caractère inutile, ni même d'utiliser beaucoup la touche Maj. En utilisant de grandes abstractions, vous pouvez écrire la première version d'un programme très rapidement. Plus tard, lorsque vous voulez optimiser, il y a un très bon profileur qui vous indiquera où concentrer votre attention.

Vous pouvez faire des boucles internes aveuglément rapidement, même en écrivant du code d'octets en ligne si vous en avez besoin.

Il y a beaucoup de bons exemples à apprendre, et le langage est suffisamment intuitif pour que vous puissiez apprendre à l'utiliser à partir d'exemples en quelques minutes. Vous n'avez pas besoin de regarder beaucoup dans le manuel. Le manuel est mince et comporte peu d'avertissements et de qualifications.

Le langage a un petit noyau et des bibliothèques puissantes et hautement orthogonales qui sont aussi soigneusement conçues que le langage de base. Les bibliothèques fonctionnent toutes bien ensemble ; tout dans le langage s'emboîte comme les pièces d'un bel appareil photo. Rien n'est obsolète ou conservé pour des raisons de compatibilité. Le code source de toutes les bibliothèques est facilement disponible. Il est facile de parler au système d'exploitation et aux applications écrites dans d'autres langages.

Le langage est construit en couches. Les abstractions de niveau supérieur sont construites de manière transparente à partir des abstractions de niveau inférieur, que vous pouvez obtenir si vous le souhaitez.

Rien ne vous est caché qui ne soit pas absolument nécessaire. Le langage n'offre des abstractions que comme un moyen de vous sauver du travail, plutôt que comme un moyen de vous dire quoi faire. En fait, le langage vous encourage à participer de manière égale à sa conception. Vous pouvez tout changer à son sujet, y compris sa syntaxe, et tout ce que vous écrivez a, autant que possible, le même statut que ce qui est prédéfini. Le langage de rêve n'est pas seulement l'open source, mais le design ouvert.

Chapitre 15

Design et Recherche

Les Américains aiment commencer une conversation en demandant « que faites-vous ? » Je n'ai jamais aimé cette question. J'ai rarement eu une réponse soignée. Mais je pense que j'ai enfin résolu le problème. Maintenant, quand quelqu'un me demande ce que je fais, je le regarde droit dans les yeux et je lui dis : « Je conçois un nouveau dialecte de Lisp. » Je recommande cette réponse à tous ceux qui n'aiment pas qu'on leur demande ce qu'ils font. La conversation se tournera immédiatement vers d'autres sujets.

Je ne me considère pas comme faisant des recherches sur les langages de programmation. J'en conçois juste un, de la même manière que quelqu'un pourrait concevoir un bâtiment, une chaise ou une nouvelle police de caractères. Je n'essaie pas de découvrir quoi que ce soit de nouveau. Je veux juste faire un langage dans lequel il sera bon de programmer.

La différence entre le design et la recherche semble être une question de nouveau par rapport au bien. Le design n'a pas besoin d'être nouveau, mais il doit être bon. La recherche n'a pas besoin d'être bonne, mais elle doit être nouvelle. Je pense que ces deux voies convergent au sommet : le meilleur signe surpassé ses prédecesseurs en utilisant de nouvelles idées, et la meilleure recherche résout des problèmes qui ne sont pas seulement nouveaux, mais qui valent la peine d'être résolus. Donc, en fin de compte, la conception et la recherche visent la même destination, en l'abordant simplement dans des directions différentes.

Que faites-vous différemment lorsque vous traitez les langages de programmation comme un problème de conception au lieu d'un sujet de recherche ?

La plus grande différence est que vous vous concentrez davantage sur l'utilisateur. Le design commence par demander à qui s'adresse-t-il et de quoi en ont-ils besoin ? Un bon architecte, par exemple, ne commence pas par créer un design qu'il impose ensuite aux utilisateurs, mais en étudiant les utilisateurs visés et en déterminant ce dont ils ont besoin.

Notez ici que j'ai dit « ce dont ils ont besoin », pas « ce qu'ils veulent ». Je ne veux pas donner l'impression que travailler en tant que designer signifie travailler comme une sorte de cuisinier à court terme, en faisant tout ce que le client vous dit de faire. Cela varie d'un domaine à l'autre dans les arts, mais je ne pense pas qu'il y ait un domaine dans lequel le meilleur travail est fait par les gens qui font exactement ce que les clients leur disent.

Le client a toujours raison en ce sens que la mesure d'une bonne conception est de savoir à quel point elle fonctionne bien pour l'utilisateur. Si vous faites un roman qui ennuie tout le monde, ou une chaise dans laquelle il est horriblement inconfortable de s'asseoir, alors vous avez fait un mauvais travail, point final. Ce n'est pas une défense de dire que le roman ou la chaise est conçu selon les principes théoriques les plus avancés.

Et pourtant, faire ce qui fonctionne pour l'utilisateur ne signifie pas simplement faire ce que l'utilisateur vous dit de faire. Les utilisateurs ne savent pas quels sont tous les choix et se trompent souvent sur ce qu'ils veulent vraiment. C'est comme être médecin. Vous ne pouvez pas simplement traiter les symptômes d'un patient. Lorsqu'un patient vous dit ses symptômes, vous devez comprendre ce qui ne va pas chez lui et traiter cela.

Cet accent mis sur l'utilisateur est une sorte d'axiome à partir duquel la majeure partie de la pratique d'une bonne conception peut être dérivée, et autour duquel la plupart des problèmes de conception sont centrés.

Quand je dis que la conception doit être pour les utilisateurs, je ne veux pas laisser entendre qu'une bonne conception vise une sorte de plus petit dénominateur commun. Vous pouvez choisir n'importe quel groupe d'utilisateurs que vous voulez. Si vous concevez un outil, par exemple, vous pouvez le concevoir pour n'importe qui, des débutants aux experts, et ce qui est un bon design pour un groupe pourrait être mauvais pour un autre. Le fait est que vous devez choisir un groupe d'utilisateurs. Je ne pense même pas que vous puissiez parler de bon ou de mauvais design, sauf en référence à un utilisateur prévu.

Vous êtes plus susceptible d'obtenir un bon design si les utilisateurs prévus incluent le concepteur lui-même. Lorsque vous concevez quelque chose pour un groupe qui ne vous inclut pas, cela a tendance à être pour les personnes

que vous considérez moins sophistiqué que vous, pas plus sophistiqué. Et regarder l'utilisateur, aussi bienveillant soit-il, semble toujours corrompre le concepteur. Je soupçonne que peu de projets de logements aux États-Unis ont été conçus par des architectes qui s'attendaient à y vivre. Vous voyez la même chose dans les langages de programmation. C, Lisp et Smalltalk ont été créés pour leurs propres concepteurs. Cobol, Ada et Java ont été créés pour que d'autres personnes les utilisent.

Si vous pensez que vous concevez quelque chose pour les idiots, il y a de fortes chances que vous ne conceviez pas quelque chose de bon, même pour les idiots.

Même si vous concevez quelque chose pour les utilisateurs les plus sophistiqués, vous concevez toujours pour les humains. C'est différent dans la recherche. En mathématiques, vous ne choisissez pas les abstractions parce qu'elles sont faciles à comprendre pour les humains ; vous choisissez celle qui rend la preuve plus courte. Je pense que c'est vrai pour les sciences en général. Les idées scientifiques ne sont pas censées être ergonomiques.

Dans les arts, les choses sont différentes. Le design est une question de peuple. Le corps humain est une chose étrange, mais lorsque vous concevez une chaise, c'est pour cela que vous concevez, et il n'y a aucun moyen de le contourner. Tous les arts doivent se plier aux intérêts et aux limites des humains. En peinture, par exemple, toutes les autres choses étant égales à une peinture avec des personnes dedans seront plus intéressantes qu'une sans. Ce n'est pas seulement un accident de l'histoire que les grandes peintures de la Renaissance soient toutes pleines de gens. S'ils ne l'avaient pas été, la peinture en tant que médium n'aurait pas le prestige qu'elle a.

Qu'on les aime ou non, les langages de programmation sont aussi pour les gens, et je soupçonne que le cerveau humain est tout aussi grumeleux et idiosyncrasique que le corps humain. Certaines idées sont faciles à saisir pour les gens et d'autres pas. Par exemple, nous semblons avoir une capacité très limitée pour traiter les détails. C'est ce fait qui fait des langages de programmation une bonne idée en premier lieu ; si nous pouvions gérer le détail, nous pourrions simplement programmer en langage machine.

Rappelez-vous également que les langages ne sont pas principalement une forme pour les programmes terminés, mais quelque chose dans lequel les programmes doivent être mis au point. N'importe qui dans les arts pourrait vous dire que vous pourriez vouloir différents moyens pour les deux situations. Le marbre, par exemple, est un moyen agréable et durable pour les idées finies, mais un moyen désespérément flexible pour développer de nouvelles idées.

Un programme, comme une preuve, est une version élaguée d'un arbre qui, dans le passé, a eu de faux départs et s'est ramifié partout. Le test d'un langage n'est donc pas simplement la propreté du programme fini, mais la propreté du chemin vers le programme terminé. Un choix de conception qui vous donne des programmes finis élégants peut ne pas vous donner un processus de conception élégant. Par exemple, j'ai écrit quelques macros qui définissent des macros qui ressemblent maintenant à de petits joyaux, mais les écrire a pris des heures d'essais et d'erreurs les plus laides, et franchement, je ne suis toujours pas tout à fait sûr qu'elles soient correctes.

Nous agissons souvent comme si le test d'une langue était la qualité des programmes finis. Cela semble tellement convaincant quand vous voyez le même programme écrit en deux langues, et qu'une version est beaucoup plus courte. Lorsque vous abordez le problème de la direction des arts, vous êtes moins susceptible de dépendre de ce type de test. Vous ne voulez pas vous retrouver avec un langage de programmation comme le marbre.

Par exemple, c'est une énorme victoire dans le développement de logiciels d'avoir un niveau supérieur interactif, ce que l'on appelle dans Lisp une boucle de lecture-eval-impression. Et quand vous en avez un, cela a des effets réels sur la conception du langage. Cela ne fonctionnerait pas bien pour un langage où vous devez déclarer des variables avant de les utiliser. Lorsque vous ne faites que taper des expressions au niveau supérieur, vous voulez être en mesure de définir x sur une certaine valeur, puis de commencer à faire des choses sur x. Vous ne voulez pas avoir à déclarer le type de x en premier. Vous pouvez contester l'un ou l'autre des locaux, mais si un langage doit avoir un niveau supérieur pour être pratique, et que les déclarations de type obligatoires sont incompatibles avec un niveau supérieur, alors aucun langage qui rend les déclarations de type obligatoires ne pourrait être pratique à programmer.

Pour obtenir un bon design, vous devez vous rapprocher, et rester proche, de vos utilisateurs. Vous devez constamment calibrer vos idées sur les utilisateurs réels. L'une des raisons pour lesquelles les romans de Jane Austen sont si bons est qu'elle les a lus à haute voix à sa famille. C'est pourquoi elle ne coule jamais dans des descriptions d'auto-indulgence et artistiques de paysages, ou de la philosophie prétentieuse. (La philosophie est là, mais elle est tissée dans l'histoire au lieu d'être collée dessus comme une étiquette.) Si vous ouvrez un roman "littéraire" moyen et que vous imaginez le lire à haute voix à vos amis comme quelque chose que vous aviez écrit, vous sentirez trop vivement à quel point ce genre de chose est imposé au lecteur.

Dans le monde des logiciels, cette idée est connue sous le nom de *Worse is Better*. En fait, il y a plusieurs idées mélangées dans le concept de *Worse is Better*, c'est pourquoi les gens se disputent toujours pour savoir si pire est en fait mieux ou non. Mais l'une des principales idées de ce mélange est que si vous construisez quelque chose de nouveau, vous devriez obtenir un prototype devant les utilisateurs dès que possible.

L'approche alternative pourrait s'appeler la stratégie du Je vous salue Marie. Au lieu de sortir un prototype rapidement et de le réaffiner progressivement, vous essayez de créer le produit complet et fini en une longue passe de touchdown. D'innombrables startups se sont détruites de cette façon pendant l'Internet Bubble. Je n'ai jamais entendu parler d'un cas où cela a fonctionné.

Ce que les gens en dehors du monde du logiciel ne réalisent peut-être pas, c'est que *Worse is Better* se trouve dans tous les arts. En dessin, par exemple, l'idée a été découverte à la Renaissance. Maintenant, presque tous les professeurs de dessin vous diront que la bonne façon d'obtenir un dessin précis n'est pas de vous frayer un chemin lentement autour du contour d'un objet, car les erreurs s'accumuleront et vous constaterez à la fin que les lignes ne se rencontrent pas. Au lieu de cela, vous devriez dessiner quelques lignes rapides à peu près au bon endroit, puis affiner progressivement ce croquis initial.

Dans la plupart des domaines, les prototypes ont traditionnellement été fabriqués à partir de différents matériaux. Les polices de caractères à couper en métal ont d'abord été conçues avec un pinceau sur papier. Les statues à couler en

bronze ont été modelées à la cire. Les motifs à broder sur les tapisseries ont été dessinés sur du papier avec un lavage à l'encre. Les bâtiments à construire en pierre ont été testés à plus petite échelle en bois.

Ce qui a rendu la peinture à l'huile si excitante, lorsqu'elle est devenue populaire pour la première fois au XVe siècle, c'est que vous pouviez faire le travail fini *à partir* du prototype. Vous pourriez faire un dessin préliminaire si vous le vouliez, mais vous n'y étiez pas tenu ; vous pouviez régler tous les détails, et même apporter des changements majeurs, au fur et à mesure que vous aviez terminé la peinture. Vous pouvez également le faire dans un logiciel. Un prototype n'a pas besoin d'être un simple modèle ; vous pouvez l'affiner en produit fini. Je pense que vous devriez toujours le faire quand vous le pouvez. Il vous permet de tirer parti des nouvelles informations que vous avez en cours de route. Mais peut-être encore plus important, c'est bon pour le moral.

Le moral est la clé du design. Je suis surpris que les gens n'en parlent pas plus. L'un de mes premiers professeurs de dessin m'a dit : si vous vous ennuyez quand vous dessinez quelque chose, le dessin aura l'air ennuyeux. Par exemple, supposons que vous deviez dessiner un bâtiment et que vous décidiez de dessiner chaque brique individuellement. Vous pouvez le faire si vous voulez, mais si vous vous ennuyez à mi-chemin et que vous commencez à fabriquer les briques au lieu d'observer chacune d'elles, le dessin aura l'air pire que si vous aviez simplement suggéré les briques.

Construire quelque chose en affinant progressivement un prototype est bon pour le moral parce qu'il vous maintient engagé. Dans les logiciels, ma règle est : ayez toujours un code de travail. Si vous écrivez quelque chose que vous pourrez tester en une heure, vous avez la perspective d'une récompense immédiate pour vous motiver. Il en va de même dans les arts, et en particulier dans la peinture à l'huile. La plupart des peintres commencent par un croquis flou et l'affinent progressivement. Si vous travaillez de cette façon, alors en principe, vous n'avez jamais à terminer la journée avec quelque chose qui semble inachevé. En effet, il y a même un dicton parmi les peintres : « Une peinture n'est jamais finie. Vous arrêtez simplement de travailler dessus. » Cette idée sera familière à tous ceux qui ont travaillé sur des logiciels.

Le moral est une autre raison pour laquelle il est difficile de concevoir quelque chose pour un utilisateur peu sophistiqué. Il est difficile de rester intéressé par quelque chose que vous n'aimez pas vous-même. Pour faire quelque chose de bien, il faut penser : « waouh, c'est vraiment génial », pas « quelle merde ; ces imbéciles vont adorer ça ».

Le design signifie faire des choses pour les humains. Mais ce n'est pas seulement l'utilisateur qui est humain. Le designer est aussi humain.

Notes

Chapitre 1, 7 à 21

1. Alberti, Leon Battista, *The Use and Abuse of Books*, traduit par Renée Watkins, Waveland Press, 1999.
2. Alors, comment réparez-vous les écoles ? La clé de la réponse peut être l'université. Lorsque vous allez dans (une bonne) université, la plupart des problèmes que je décris sont résolus. La solution peut donc venir de la question suivante : comment faire en sorte que la vie des nerds adolescents ressemble davantage à la vie universitaire ?
L'enseignement à domicile offre une solution immédiate, mais ce n'est probablement pas la solution optimale. Pourquoi les parents n'éduquent-ils pas leurs enfants à la maison tout au long de l'université ? Parce que l'université offre des opportunités, l'enseignement à domicile ne peut pas se doubler ? Il en serait de même pour le lycée si c'était bien fait.

Chapitre 2, 22 à 38

1. Johnson a écrit dans la préface de son Shakespeare :
« Il a longtemps survécu à son siècle, le terme communément fixé comme le test du mérite littéraire. Quels que soient les avantages qu'il pouvait autrefois tirer des allusions personnelles, des coutumes locales ou des opinions temporaires, ont été perdus pendant de nombreuses années ; et tout le choix de la gaieté ou du motif du chagrin, que les modes de vie artificielle lui offraient, n'obscurcissent maintenant que les scènes qu'ils éclairaient autrefois. Les effets de la faveur et de la concurrence sont à leur fin ; la tradition de ses amitiés et de ses inimitiés a péri ; ses œuvres ne soutiennent aucune opinion avec des arguments, ni ne fournissent à aucune faction des invectives ; elles ne peuvent se livrer à la vanité ni à la malignité satisfaite, mais sont lues sans autre raison que le désir de plaisir, et ne sont donc louées que lorsque le plaisir est obtenu...»
2. La pire chose que la photographie a faite à la peinture a peut-être été de tuer le meilleur travail de jour. La plupart des grands peintres de l'histoire se sont soutenus en peignant des portraits. Peu après l'invention de la

photographie, ils ont été concurrencés par des hackers qui travaillaient à partir de photographies. (Cette méthode est également plus facile pour la gardienne.) La classe de peintres techniquelement qualifiés a alors plus ou moins disparu, et le rôle de la compétence dans le prix de la peinture a été remplacé par la marque (qui dépend également grandement de la photographie, ou, plus précisément, des photographies reproduites dans des livres et des magazines).

3. Microsoft décourage les employés de contribuer à des projets open source, même pendant leur temps libre. Mais tant des meilleurs hackers informatiques travaillent sur des projets open source maintenant que le principal effet de cette politique peut être de leur rendre difficile l'embauche de programmeurs de premier ordre.
4. Ce que vous apprenez sur la programmation à l'université est comme ce que vous apprenez sur les livres ou vêtements : quel mauvais goût vous aviez au lycée.
5. Voici un exemple d'empathie appliquée. Chez Viaweb, si nous ne pouvions pas décider d'entre deux alternatives, nous nous demanderions, qu'est-ce que nos concurrents détesteraient le plus ? À un moment donné, un concurrent a ajouté une fonctionnalité à son logiciel qui était fondamentalement moins utilisée, mais comme c'était l'une des rares qu'ils avaient que nous n'avions pas, ils en ont fait beaucoup dans la presse commerciale. Nous aurions pu essayer d'expliquer que la fonctionnalité était inutile, mais nous avons décidé qu'elle ennuierait davantage notre concurrent si nous la mettions en œuvre nous-mêmes, alors nous avons bidouillé notre propre version cet après-midi-là.
6. À l'exception des éditeurs de texte et des compilateurs. Les hackers n'ont pas besoin d'empathie pour les concevoir, car ils sont eux-mêmes des utilisateurs typiques.
7. Eh bien, presque. Ils ont quelque peu dépassé la RAM disponible, provoquant beaucoup d'échange de disques, mais cela pourrait être résolu en achetant un lecteur de disque supplémentaire.

8. Abelson, Harold et Gerald Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, 1985.
9. La façon de rendre les programmes faciles à lire est de ne pas les remplir de commentaires. Je prendrais la règle d'Abelson et Sussman un peu plus loin. Les langages de programmation doivent être conçus pour exprimer des algorithmes, et seulement accessoirement pour indiquer aux ordinateurs comment les exécuter. Un bon langage de programmation devrait être meilleur pour expliquer les logiciels que l'anglais. Vous ne devriez avoir besoin de commentaires que lorsqu'il y a une sorte de kludge dont vous devez avertir les lecteurs, tout comme sur une route, il n'y a que des flèches sur des parties aux courbes étonnamment nettes.

Chapitre 3, 39 à 55

1. L'Inquisition n'a probablement jamais eu l'intention d'exécuter sa menace de torture. Mais c'est parce que Galilée a clairement indiqué qu'il ferait tout ce qu'ils demandaient. S'il avait refusé, ils n'auraient pas simplement reculé. Peu de temps avant qu'ils n'aient brûlé le philosophe Giordano Bruno lorsqu'il s'est montré intransigeant.
2. De nombreuses organisations publient des listes de ce que vous ne pouvez pas dire en leur sein. Malheureusement, ces listes sont généralement à la fois incomplètes, parce qu'il y a des choses si choquantes qu'elles ne s'attendent même pas à ce que quelqu'un les dise, et en même temps si générales qu'elles ne pourraient pas être appliquées littéralement. C'est un code de discours d'université rare qui n'interdirait pas, pris au pied de la lettre, à Shakespeare.
3. Kundel, H.L., C.F. Nodine, et E.A. Krupinski, « Searching for lung nodules: Visual dwell indicates locations of false-positive and false-negative decisions », *Investigative Radiology*, 24 (1989), 472-478.
4. Le verbe "à différer" est du jargon informatique, mais c'est le seul mot avec exactement le sens que je veux. Voir le glossaire.

5. Il peut sembler à partir de cela que je suis une sorte de relativiste moral. Loin de ça. Je pense que le "jugement" est l'une des étiquettes utilisées à notre époque pour supprimer la discussion, et que nos tentatives d'être "sans jugement" sembleront à l'avenir l'une des choses les plus comiques à notre sujet.
6. Cela rend le monde confus pour les enfants, car ce qu'ils voient n'est pas en désaccord avec ce qu'on leur dit. Je n'ai jamais pu comprendre pourquoi, par exemple, les portugais "explorateurs" avaient commencé à se frayer un chemin le long de la côte africaine. En fait, ils étaient à la recherche d'esclaves.

De Azurara, Gomes Eannes, *Chronicle of the Discovery of Guinea*, dans Almeida (ed.), *Conquests and Discoveries of Henry the Navigator*, George Allen & Unwin, 1936.
7. Les enfants apprennent bientôt ces mots de leurs amis, mais ils savent qu'ils ne sont pas censés les utiliser. Donc, pendant un certain temps, vous avez un état de choses comme quelque chose d'une comédie musicale, où les parents utilisent ces mots parmi leurs pairs, mais jamais devant les enfants, et les enfants utilisent les mots parmi leurs pairs, mais jamais devant leurs parents.
8. Le logo de Viaweb était un cercle rouge uni avec un V blanc au milieu. Après l'avoir utilisé pendant un certain temps, je me souviens avoir pensé, vous savez, que c'est un symbole vraiment plein, un cercle rouge. Le rouge est sans doute la couleur la plus basique, et le cercle la forme la plus basique. Ensemble, ils ont eu un tel coup de poing visuel. Pourquoi plus d'entreprises américaines n'ont-elles pas un cercle rouge comme logo ? Ahh, oui...
9. La peur est de loin la plus forte des deux forces. Parfois, quand j'entends quelqu'un utiliser le mot "gyp", je lui dis, avec une expression sérieuse, qu'on ne peut plus utiliser ce mot parce qu'il est considéré comme dénigrant pour les Romani (alias Gypsies). En fait, les dictionnaires ne sont pas d'accord sur son étymologie. Mais la réaction à cette blague est presque toujours une réaction de conformité légèrement terrifiée. Il y a quelque chose dans la mode, dans les vêtements ou les idées, qui enlève

la confiance des gens : quand ils apprennent quelque chose de nouveau, ils pensent que c'est quelque chose qu'ils auraient déjà dû savoir.

10. C'est le seul exemple de sujet dans cet essai de quelque chose que vous ne pouvez pas dire. Il est le principal tabou de la vie universitaire. Au sein des universités, c'est un axiome tacite selon lequel tous les domaines d'études sont intellectuellement égaux. Il ne fait aucun doute que cet axiome aide les choses à mieux fonctionner. Mais si vous considérez à quel point il faudrait une monnaie étonnante pour qu'elle soit vraie, et à quel point il serait pratique pour tout le monde de la traiter comme vraie même si ce n'était pas le cas, comment pouvez-vous ne pas la remettre en question ?

Surtout lorsque vous considérez certains des corollaires, cela vous oblige à accepter. Par exemple, cela signifierait qu'il ne pourrait pas y avoir de hauts et de bas dans un domaine individuel. À moins que tous les champs ne oscillent de manière synchronisée. (Vous devez vraiment vous étirer pour sauver celui-ci.)

Et puis, que faites-vous des universités qui ont des départements comme les arts culinaires ou la gestion du sport ? Si vous acceptez cet axiome, jusqu'où s'étend-il ? Voulez-vous vraiment vous retrouver à défendre la position dont la géométrie différentielle n'est pas plus difficile que la cuisine ?

11. Vraisemblablement, au sein de l'industrie, de telles pensées seraient considérées comme « négatives ». Un autre label, un peu comme « défaitiste ». Peu importe, il faut se demander, sont-ils vrais ou non ? En effet, la mesure d'une organisation saine est probablement le degré auquel les pensées négatives sont autorisées. Dans les endroits où un excellent travail est fait, l'attitude semble généralement critique et sarcastique plutôt que « positive » et « favorable ». Les gens que je connais qui font un excellent travail pensent qu'ils sont nuls, mais que tous les autres le sont plus.

12. Behar, Richard, "The Thriving Cult of Greed and Power", *Time*, 6 mai 1991.

13. Healy, Patrick, "Summers hits 'anti-Semitic' actions", *Boston Globe*, 20 Septembre 2002.
14. « Tinkerers' champion », *The Economist*, 20 juin 2002.
15. Par là, je veux dire que vous devez devenir un controversialiste professionnel, pas que les opinions de Noam Chomsky = ce que vous ne pouvez pas dire. Si vous disiez réellement les choses que vous ne pouvez pas dire, vous choqueriez les conservateurs et les libéraux de la même manière - tout comme, si vous retourniez à l'Angleterre victorienne dans une machine à remonter le temps, vos idées choqueraient les whigs et les conservateurs de la même manière.
16. Traub, James, « Harvard Radical », *New York Times Magazine*, 24 août 2003.
17. Miller, Arthur, *The Crucible in History and Other Essays*, Methuen, 2000.
18. Certains évitent les "mauvais" en tant que jugements, et à la place d'utiliser des euphémismes plus neutres comme "négatifs" ou "destructeurs".

Chapitre 4, 56 à 62

1. J'ai prévu d'apprendre à choisir des verrous aussi. Mais pas juste à l'extérieur de la curiosité intellectuelle. Lorsque j'étais à peu près à mi-chemin de l'école supérieure, le corps intelligent mais truculent de hackers de premier cycle qui avaient l'habitude d'administrer tous les ordinateurs a été remplacé par un administrateur système professionnel qui avait l'habitude de rentrer chez lui à 5 heures et de laisser la porte de la salle des machines verrouillée. Si un ordinateur était coincé, on s'attendait à ce qu'il attende jusqu'au matin pour le redémarrer. Un plan complètement peu pratique, car à cette époque, nous n'avons souvent même pas commencé à travailler avant 17 heures. Heureusement, à Aiken Lab (depuis démolie), il y avait un espace entre les étages et une trappe juste au-dessus du bureau de l'administrateur système. Lorsque nous

avions besoin de la clé de la salle des machines, nous tombions par le plafond et la sortions du tiroir de son bureau.

Une nuit, vers 3 heures du matin, alors que je descendais sur le bureau de l'administrateur système, des alarmes auriculaires se sont déclenchées dans tout le bâtiment. « Putain », ai-je pensé (désolé pour le blasphème, mais je me souviens clairement d'avoir pensé cela), « ils ont câblé l'endroit. » Je suis sorti de ce bâtiment en une trentaine de secondes. Je me suis précipité à la maison (par une tempête de pluie diluvienne), essayant d'avoir l'air nonchalant, mais à ma conscience coupable, chaque voiture ressemblait à une Crown Victoria. Lorsque je me suis présenté au laboratoire le lendemain, j'étais déjà en train de répéter ma défense, mais il n'y avait pas d'e-mail inquiétant qui m'attendait. Il s'est avéré que les alarmes avaient été déclenchées par la foudre pendant la tempête.

2. Ce n'est pas seulement le contenu des produits qui est de plus en plus un logiciel. Au fur et à mesure que la fabrication devient de plus en plus automatisée, les conceptions deviennent également des logiciels.
3. Je serais heureux de donner mon nom pour cette courbe. Le fait de donner un nom à une idée sera mieux acceptée.

Chapitre 5, 63 à 96

1. Réalisant que beaucoup d'argent est dans les services, les entreprises qui construisent des clients légers ont généralement essayé de combiner le matériel avec un service en ligne. Cette approche n'a pas bien fonctionné, en partie parce que vous avez besoin de deux types d'entreprises différentes pour construire des produits électroniques grand public et gérer un service en ligne, et en partie parce que les utilisateurs détestent l'idée. Donner la poignée et faire de l'argent sur les lames peut fonctionner pour Gillette, mais un rasoir est un engagement moins important qu'un terminal web.

Les fabricants de téléphones portables sont satisfaits de vendre du matériel sans essayer de plafonner également les revenus du service. Cela devrait probablement aussi être le modèle pour les clients Internet. Si quelqu'un venait de vendre une jolie petite boîte avec un navigateur Web

que vous pourriez utiliser pour vous connecter via n'importe quel FAI, chaque technophobe du parc en achèterait une.

2. La sécurité dépend toujours plus de la volonté de ne pas se tromper que de n'importe quelle décision de conception mais la nature des logiciels basés sur des serveurs incitera les développeurs à accorder plus d'attention au fait de ne pas se tromper. La compromission d'un serveur pourrait causer de tels dommages que les ASP (qui veulent rester en activité) sont susceptibles de faire attention à la sécurité.
3. En 1995, lorsque nous avons lancé Viaweb, les applets Java étaient censés être la technologie que tout le monde allait utiliser pour développer des applications basées sur le serveur. Les applets nous semblaient une idée à l'ancienne. Télécharger des programmes à exécuter sur le client ? Plus simple, il suffit d'aller jusqu'au bout et d'exécuter les programmes sur le serveur. Nous avons perdu peu de temps sur les applets, mais d'innombrables autres startups ont dû être attirées dans cette fosse de goudron. Peu semblent s'en être échappés vivants.
4. Ce point est dû à Trevor Blackwell, qui ajoute : « Le coût des logiciels d'écriture augmente plus que linéairement avec sa taille. Peut-être est-ce principalement dû à la correction d'anciens bugs, et le coût peut être plus linéaire si tous les bugs sont trouvés rapidement. »
5. Le type de bugs le plus difficile à trouver peut être une variante du bogue composé où un bogue compense un autre. Lorsque vous corrigez un bug, l'autre devient visible. Mais il semblera que la solution soit en faute, puisque c'est la dernière chose que vous avez changée.
6. Au sein de Viaweb, nous avons déjà eu un concours pour décrire la pire chose à propos de notre logiciel. Deux personnes du support client sont à égalité pour le premier prix avec des entrées dont je frissonne encore de me rappeler. Nous avons résolu les deux problèmes immédiatement.
7. Robert Morris a écrit le système de commande, que les acheteurs utilisaient pour placer des commandes. Trevor Blackwell a écrit le générateur d'images et le gestionnaire, que les commerçants utilisaient

pour récupérer les commandes, afficher les statistiques, configurer les noms de domaine, etc. J'ai écrit à l'éditeur, que les commerçants utilisaient pour construire leurs sites. Le système de commande et le générateur d'images ont été écrits en C et C++, le gestionnaire principalement en Perl et l'éditeur en Common Lisp.

8. J'utilise « exponentiellement » au sens familier ici. Correctement, il devrait être « polynomialement ».
9. La discrimination par les prix est si répandue que j'ai été surpris de constater qu'elle a été interdite aux États-Unis par la loi Robinson-Patman de 1936. Cette loi ne semble pas être vigoureusement appliquée.
10. Dans *No Logo*, Naomi Klein dit que les marques de vêtements favorisées par les « jeunes urbains » n'essaient pas trop d'empêcher le vol à l'étalage parce que dans leur marché cible, les voleurs à l'étalage sont également les leaders de la mode.
11. Les entreprises se demandent souvent ce qu'elles doivent externaliser et ce qu'elles ne doivent pas faire. Une réponse possible : externaliser tout travail qui n'est pas directement exposé à la pression concurrentielle, car l'externalisation l'exposera ainsi à la pression concurrentielle. (Je veux dire « externaliser » dans le sens d'embaucher une autre entreprise pour le faire, et non dans le sens plus spécifique de l'embauche d'une entreprise étrangère.)
12. Les deux gars étaient Dan Bricklin et Bob Frankston. Dan a écrit un prototype en Basic en quelques jours, puis au cours de l'année suivante, ils ont travaillé ensemble (la plupart du temps la nuit) pour faire une version plus puissante écrite en langage machine 6502. Dan était à la Harvard Business School à l'époque et Bob avait nominalement un travail de jour pour écrire un logiciel. « Il n'y avait pas de grand risque à faire des affaires », m'a dit Bob. « S'il a échoué, il a échoué. Ce n'est pas grand-chose. »
13. Ce n'est pas aussi facile que je le fais paraître. Il a fallu beaucoup de temps pour que le bouche à oreille se mette en marche, et nous n'avons

pas eu beaucoup de couverture médiatique jusqu'à ce que nous embauchions Schwartz Communications, probablement la meilleure entreprise de relations publiques de haute technologie de l'entreprise, pour 16 000 \$/mois (plus quelques mandats). Cependant, il était vrai que le seul canal important était notre propre site web.

14. Si le Mac était si génial, pourquoi a-t-il perdu ? Coût, encore une fois. Microsoft s'est concentré sur le secteur des logiciels et a déclenché un essaim de fournisseurs de composants bon marché sur le matériel Apple. Cela n'a pas aidé non plus que les costumes aient pris le relais pendant une période critique. (Et il n'a pas encore perdu. Si Apple devait transformer l'iPod en un téléphone portable avec un navigateur Web, Microsoft aurait de gros ennuis.)

15. Une chose qui aiderait les applications basées sur le Web, et aiderait à l'autre génération de logiciels d'être éclipsée par Microsoft, serait un bon navigateur open source. Un petit navigateur rapide serait une bonne chose en soi, et encouragerait les entreprises à construire de petits appareils Web.

Mieux que ce soit, un bon navigateur open source pourrait faire évoluer HTTP et HTML (comme par ex. Perl a). Vous vous souvenez quand chaque version de Netscape ajoutait de nouvelles fonctionnalités au HTML ? Pourquoi cela a-t-il été arrêté ?

Cela aiderait grandement les applications Web à faire la distinction entre la sélection d'un lien et son suivi ; tout ce dont vous auriez besoin pour le faire serait une amélioration triviale de HTTP, pour permettre plusieurs URL dans une demande. Les menus en cascade seraient également bons. Si vous voulez changer le monde, écrivez une nouvelle mosaïque. Vous pensez qu'il est trop tard ? En 1998, beaucoup de gens pensaient qu'il était trop tard pour lancer un nouveau moteur de recherche, mais Google leur a prouvé qu'ils avaient tort. Il y a toujours de la place pour quelque chose de nouveau si c'est beaucoup mieux.

16. Trevor Blackwell, qui en sait probablement plus à ce sujet par expérience personnelle que quiconque, écrit :

« J'irais plus loin en disant que parce que les logiciels sur serveur sont si durs pour les programmeurs, cela provoque un changement économique fondamental par rapport aux grandes grandes grandes. Cela nécessite le genre d'intensité et de dévouement de la part des programmeurs qu'ils ne seront prêts à fournir que lorsqu'il s'agit de leur propre entreprise. Les entreprises de logiciels peuvent embaucher des personnes qualifiées pour travailler dans un environnement pas trop exigeant, et peuvent embaucher des personnes non qualifiées pour supporter des difficultés, mais elles ne peuvent pas embaucher des personnes hautement qualifiées pour se casser le cul. Étant donné que le capital n'est plus nécessaire, les grandes sociétés n'ont pas grand-chose à apporter à la table. »

17.Je n'utiliserais même pas Javascript, si j'étais vous ; Viaweb ne l'a pas fait. La plupart du Javascript que je vois sur le Web n'est pas nécessaire, et une grande partie se casse. Et quand vous commencez à pouvoir parcourir des pages Web réelles sur votre téléphone portable ou votre PDA (ou votre grille-pain), qui sait s'ils le tiendront même en charge ?

Chapitre 6, 97 à 120

1. Une chose précieuse que vous avez tendance à obtenir uniquement dans les startups est la stabilité ininterrompue. Les différents types de travail ont des quanta de temps différents. Quelqu'un qui relit un script de manuscrit pourrait probablement être interrompu toutes les quinze minutes avec peu de perte de productivité. Mais le temps quantique pour le bidouillage est très long : cela pourrait prendre une heure juste pour charger un problème dans votre tête. Ainsi, le coût d'avoir quelqu'un du personnel qui vous appelle au sujet d'un formulaire que vous avez oublié de remplir peut être énorme.

C'est pourquoi les hackers vous donnent un regard si fou alors qu'ils se tournent de leur écran pour répondre à votre question. À l'intérieur de leur tête, un gigantesque château de cartes vacilla.

La simple possibilité d'être interrompu dissuade les hackers de se lancer des projets difficiles. C'est pourquoi ils ont tendance à travailler tard dans la nuit, et pourquoi il est possible d'écrire d'excellents logiciels dans une cabine (sauf tard dans la nuit).

Un grand avantage des startups est qu'elles n'ont pas encore de personnes qui vous interrompent. Il n'y a pas de service du personnel, et donc pas de formulaire ni personne pour vous appeler à ce sujet.

2. Face à l'idée que les gens qui travaillent pour des start-ups pourraient être 20 ou 30 fois plus productifs que ceux qui travaillent pour de grandes entreprises, les dirigeants des grandes entreprises se demanderont naturellement comment pourrais-je faire en quoi les gens qui travaillent pour moi le fassent ? La réponse est simple : payez-les.

En interne, la plupart des entreprises sont gérées comme des États communistes. Si vous croyez aux marchés libres, pourquoi ne pas transformer votre entreprise en un seul marché ?

Hypothèse : Une entreprise sera au maximum rentable lorsque chaque employé sera payé proportionnellement à la richesse qu'il génère.

3. Jusqu'à récemment, même les gouvernements ne comprenaient parfois pas la distinction entre l'argent et la richesse. Adam Smith (*Wealth of Nations*, v:i) en mentionne plusieurs qui ont essayé de préserver leur « richesse » en interdisant l'exportation d'or ou d'argent. Mais avoir plus de moyens d'échange ne rendrait pas un pays plus riche ; si vous avez plus d'argent à la recherche de la même quantité de richesse matérielle, le seul résultat est des prix plus élevés.

4. Il y a beaucoup de sens du mot « richesse », qui ne sont pas tous matériels. Je n'essaie pas de faire un point philosophique profond ici sur ce qui est le vrai genre. J'écris à propos d'un sens spécifique et plutôt technique du mot « richesse ». Ce pour quoi les gens vous donneront de l'argent. C'est une sorte de richesse intéressante à étudier, parce que c'est le genre qui vous empêche de mourir de faim. Et ce pour quoi les gens vous donneront de l'argent dépend d'eux, pas de vous.

Lorsque vous démarrez une entreprise, il est facile de penser que les clients veulent ce que vous faites. Pendant l'Internet Bubble, j'ai parlé à une femme qui, parce qu'elle aimait le plein air, commençait un "portail extérieur". Vous savez quel genre d'entreprise vous devriez démarrer si vous aimez le plein air ? Un pour récupérer des données à partir de disques durs plantés.

Quel est le lien ? Aucun du tout. Ce qui est précisément ce que je veux dire. Si vous voulez créer de la richesse (dans le sens technique étroit de ne pas mourir de faim), alors vous devriez être particulièrement sceptique quant à tout plan qui se concentre sur les choses que vous aimez faire. C'est là que votre idée de ce qui est précieux est la moins susceptible de coïncider avec celles d'autres gens.

5. Dans l'arrêt moyen, vous rendrez probablement plus de plus pauvres les microscopiques, en causant une petite quantité de dommages à l'environnement. Bien que les coûts environnementaux doivent être pris en compte, ils ne font pas de la richesse un jeu à somme nulle. Par exemple, si vous réparez une machine qui est cassée parce qu'une pièce a été dévissée, vous créez de la richesse sans coût environnemental.
6. Beaucoup de gens se sentent confus et déprimés au début de la vingtaine. La vie semblait tellement plus amusante à l'université. Eh bien, bien sûr que c'était le cas. Ne vous laissez pas berner par les similitudes de surface. Vous êtes passé d'invité à serviteur. Il est possible de s'amuser dans ce nouveau monde. Entre autres choses, vous pouvez maintenant aller derrière les portes qui disent "personnel autorisé uniquement". Mais le changement est d'abord un choc, et c'est d'autant plus grave si vous n'en êtes pas conscient.
7. Lorsque les VC nous ont demandé combien de temps il faudrait à une autre startup pour dupliquer notre logiciel, nous avions l'habitude de répondre qu'ils ne seraient probablement pas en mesure de le faire du tout. Je pense que cela nous a fait paraître naïfs ou menteurs.
8. Peu de technologies ont un inventeur clair. Donc, en règle générale, si vous connaissez l'"inventeur" de quelque chose (le téléphone, la chaîne de montage, l'avion, l'ampoule, le transistor), c'est parce que leur entreprise en a gagné de l'argent, et que les gens de relations publiques de l'entreprise ont travaillé dur pour répandre l'histoire. Si vous ne savez pas qui a inventé quelque chose (l'automobile, la télévision, l'ordinateur, le moteur à réaction, le laser), c'est parce que d'autres entreprises ont gagné tout l'argent.

9. C'est un bon plan pour la vie en général. Si vous avez deux choix, choisissez le plus difficile. Si vous essayez de décider de sortir courir ou de rester à la maison et de regarder la télévision, allez courir. La raison pour laquelle cette astuce fonctionne si bien est probablement que lorsque vous avez deux choix et que l'un est plus difficile, la seule raison pour laquelle vous considérez même l'autre est la paresse. Vous savez au fond de votre esprit quelle est la bonne chose à faire, et cette astuce vous oblige simplement à le reconnaître.
10. Ce n'est sans doute pas un hasard si la classe moyenne est d'abord apparue en Italie du Nord et dans les petits pays, où il n'y avait pas de gouvernement central fort. Ces deux régions étaient les plus riches de leur époque et sont devenues les centres jumeaux à partir desquels la civilisation de la Renaissance a rayonné. Si elles ne jouent plus ce rôle, c'est parce que d'autres endroits, comme les États-Unis, ont été plus fidèles aux principes qu'ils ont découverts.
11. Cela peut en effet être une condition suffisante. Mais si oui, pourquoi la révolution industrielle n'a-t-elle pas eu lieu plus tôt ? Deux réponses possibles (et non incompatibles) : (a) Elle l'a fait. La révolution industrielle faisait partie d'une série. (b) Parce que dans les villes médiévales, les monopoles et les règlements de guilde ont initialement ralenti le développement de nouveaux moyens de production.

Chapitre 7, 121 à 133

1. Une partie de la raison pour laquelle ce sujet est litigieux est que certains des autres sujets sur le sujet de la richesse - étudiants universitaires, héritiers, professeurs, politiciens et journalistes - ont le moins d'expérience dans sa création. (Ce phénomène sera familier à tous ceux qui ont entendu des conversations sur le sport dans un bar.)
La plupart des étudiants sont pour la plupart toujours à la charge de leurs parents et ne se sont pas arrêtés pour réfléchir à l'origine de cet argent. Les héritiers seront sur le don parental à vie. Les professeurs et les politiciens vivent dans les tourbillons socialistes de l'économie, à l'écart de la création de richesse, et sont payés à un taux fixe, quelle que soit l'intensité à laquelle ils travaillent. Et les journalistes, dans le cadre de

leur code professionnel, se séparent de la moitié des entreprises pour lesquelles ils travaillent (le département des ventes publicitaires). Beaucoup de ces personnes ne se heurtent jamais au fait que l'argent qu'elles reçoivent représente la richesse - une richesse que, sauf dans le cas des journalistes, quelqu'un d'autre a créée plus tôt. Ils vivent dans un monde où le revenu est distribué par une autorité centrale selon une notion abstraite d'équité (ou au hasard, dans le cas des héritiers), plutôt que donné par d'autres personnes en échange de quelque chose qu'ils voulaient, de sorte qu'il peut leur sembler injuste que les choses ne fonctionnent pas de la même manière dans le reste de l'économie. (Certains professeurs créent beaucoup de richesse pour la société. Mais l'argent qu'ils sont payés n'est pas une contrepartie. C'est plus dans la nature d'un investissement.)

2. Lorsqu'on lit les origines de la Fabian Society, on a l'impression qu'il s'agit d'un projet concocté par les enfants-héros édouardiens d'Edith Nesbit, *The Wouldbegoods*.
3. Selon une étude de la Corporate Library, la rémunération totale médiane, y compris le salaire, les primes, les subventions d'actions et l'exercice des options d'achat d'actions, des PDG de S&P 500 en 2002 était de 3,65 millions de dollars. Selon Sports Illustrated, le salaire moyen du joueur de la NBA au cours de la saison 2002-2003 était de 4,54 millions de dollars, et le salaire moyen du joueur de baseball de la ligue majeure au début de la saison 2003 était de 2,56 millions de dollars. Selon le Bureau of Labor Statistics, le salaire annuel moyen aux États-Unis en 2002 était de 35 560 \$.
4. Au début de l'empire, le prix d'un esclave adulte ordinaire semble avoir été d'environ 2 000 sesterces (ex. Horace, *Sat.* II.7.43). Une servante a coûté 600 (Martial VI.66), tandis que Columella (III.3.8) dit qu'un viticulteur qualifié valait 8 000 \$. Un médecin, P. Decimus Eros Merula, a payé 50 000 sesterces pour sa liberté (Dessau, *Inscriptiones* 7812). Sénèque (*Ep*, XXVII.7) rapporte qu'un Calvisius Sabinus a payé 100 000 sesterces chacun pour des esclaves ayant appris des classiques grecs. Pline (*Hist. Nat.* VII.39) dit que le prix le plus élevé payé pour un esclave jusqu'à son époque était de 700 000 sesterces, pour le linguiste (et

probablement enseignant) Daphnis, mais que cela avait depuis été dépassé par les acteurs qui achetaient leur propre liberté.

L'Athènes classique a connu une variation similaire des prix. Un ouvrier ordinaire valait environ 125 à 150 drachmes. Xénophon (*Mem. II.5*) mentionne que les prix ont augmenté de 50 à 6 000 drachmes (pour le directeur d'une mine d'argent).

Pour en savoir plus sur l'économie de l'esclavage ancien, voir :

Jones, A. H. M., "L'esclavage dans le monde antique", *Economic History Review*, 2:9 (1956), 185-199, réimprimé dans Finley, M. I. (éd.), *L'esclavage dans l'Antiquité classique*, Heffer, 1964.

5. Ératosthène (276-195 av. J.-C.) a utilisé des longueurs d'ombre dans différentes villes pour estimer la circonférence de la Terre. Il n'était en congé que d'environ 2 %.
6. Non, et Windows, respectivement.
7. L'une des plus grandes divergences entre le Daddy Model et la réalité est l'évaluation du travail acharné. Dans le Daddy Model, le travail acharné est en soi méritant. En réalité, la richesse se mesure à ce que l'on livre, et non à l'effort qu'elle coûte. Si je peins la maison de quelqu'un, le propriétaire ne devrait pas me payer de supplément pour le faire avec une brosse à dents.
Il semblera à quelqu'un qui opère encore implicitement sur le Daddy Model qu'il est injuste que quelqu'un travaille dur et ne soit pas beaucoup payé. Pour aider à clarifier la question, débarrassez-vous de tout le monde et mettez votre travailleur sur une île déserte, en chassant et en ramassant des fruits. S'il est mauvais dans ce cas, il travaillera très dur et ne se retrouvera pas avec beaucoup de nourriture. Est-ce injuste ? Qui est injuste envers lui ?
8. Une partie de la raison de la rareté du Daddy Model peut-être le double sens de « distribution ». Lorsque les économistes parlent de « distribution du revenu », ils veulent dire distribution statistique. Mais lorsque vous utilisez fréquemment l'expression, vous ne pouvez pas vous empêcher de l'associer à l'autre sens du mot (comme dans par exemple "distribution de l'aumône"), et donc de voir inconsciemment la richesse comme quelque

chose qui coule d'un robinet central. Le mot « régressif » tel qu'il est appliqué aux taux d'imposition a un effet similaire, du moins sur moi ; comment quelque chose de régressif peut-il être bon ?

9. « Dès le début du règne de Thomas Lord Roos était un courtier assidu du jeune Henri VIII et allait bientôt en récolter les fruits. En 1525, il a été fait chevalier de la Jarretière et a reçu le comte de Rutland. Dans les années trente, son soutien à la brèche avec Rome, son zèle pour écraser le Pèlerinage de la Grâce et sa volonté de voter la peine de mort dans la succession de procès de trahison spectaculaires qui ont ponctué les progrès matrimoniaux erratiques d'Henry ont fait de lui un candidat évident pour l'octroi de biens monastiques. »
Stone, Lawrence, *Family and Fortune : Studies in Aristocratic Finance in the XVIe et XVIIe siècles*, Oxford University Press, 1973, p. 166.
10. Il y a des preuves archéologiques de grands établissements plus tôt, mais il est difficile de dire ce qui se passait en eux.
Hodges, Richard et David Whitehouse, *Mohammed, Charlemagne et les Origines de l'Europe*, Cornell University Press, 1983.
11. William Cecil et son fils Robert ont été tour à tour le ministre le plus puissant de la couronne et tous deux ont profité de leur position pour amasser des fortunes parmi les plus importantes de leur époque. Robert en particulier a porté la corruption jusqu'à la trahison. « En tant que secrétaire d'État et principal conseiller du roi Jacques en matière de politique étrangère, [il] a été un récipiendaire spécial de faveurs, se voyant offrir de gros pots-de-vin par les Néerlandais pour ne pas faire la paix avec l'Espagne, et de gros pots-de-vin par l'Espagne pour faire la paix. » (Stone, *op. cit.*, p. 17.)
12. Bien que Balzac ait fait beaucoup d'argent de l'écriture, il était notoirement imprévoyant et a été accablé par des dettes toute sa vie.
13. Une Timex gagnera ou perdra approximativement .5 par jour. La montre mécanique la plus précise, la Patek Philippe 10 Day Tourbillon, est évaluée à -1,5 à +2 secondes. Son prix de détail est d'environ 220 000 \$.

14. Si on lui demandait de choisir laquelle était la plus chère, une limousine de dix passagers Lincoln Town Car 1989 bien préservée (5 000 \$) ou une berline Mercedes S600 2004 (12 000 \$), l'Édouardien moyen pourrait bien se tromper.
15. Pour dire quoi que ce soit de significatif sur les tendances des revenus, vous devez parler du revenu réel, ou du revenu tel que mesuré dans ce qu'il peut acheter. Mais la façon habituelle de calculer le revenu réel ignore une grande partie de la croissance de la richesse au fil du temps, car elle dépend d'un indice des prix à la consommation créé par boulonner de bout en bout une série de chiffres qui ne sont que locaux précis, et qui n'incluent pas les prix des nouvelles inventions jusqu'à ce qu'elles deviennent si courantes que leurs prix se stabilisent.
- Donc, bien que nous puissions penser qu'il était beaucoup mieux de vivre dans un monde avec des antibiotiques ou des voyages aériens ou un réseau électrique que sans, les statistiques de revenu réel calculées de la manière habituelle nous prouveront que nous ne sommes que légèrement plus riches pour avoir ces choses.
- Une autre approche serait de demander, si vous retourniez à l'année x dans une machine à remonter le temps, combien auriez-vous à dépenser en biens commerciaux pour faire fortune ? Par exemple, si vous reveniez à 1970, ce serait certainement moins de 500 \$, parce que la puissance de traitement que vous pouvez obtenir pour 500 \$ aujourd'hui aurait valu au moins 150 millions de dollars en 1970. La fonction devient asymptotique assez rapidement, parce que pendant plus d'une centaine d'années environ, vous pourriez obtenir tout ce dont vous avez besoin dans les poubelles d'aujourd'hui. En 1800, une bouteille de boisson en plastique vide avec un bouchon à vis aurait semblé un miracle de fabrication humaine.
16. Certains diront que cela équivaut à la même chose, parce que les riches ont de meilleures possibilités d'éducation. C'est un point valable. Il est toujours possible, dans une certaine mesure, "d'acheter" le chemin de vos enfants dans les meilleures universités en les envoyant dans des écoles privées qui, en fait, piratent le processus d'admission à l'université. Selon un rapport de 2002 du National Center for Education Statistics, environ 1,7 % des enfants américains fréquentent des écoles privées et

non sectaires. À Princeton, 36 % de la classe de 2007 provenait de ces écoles. (Intéressant, le nombre à Harvard est nettement inférieur, soit environ 28 %.) De toute évidence, il s'agit d'un énorme trou de boucle. Il semble au moins se fermer, pas s'élargir.

Peut-être que les concepteurs des processus d'admission devraient tirer une leçon de l'exemple de la sécurité informatique, et au lieu de simplement supposer que leur système ne peut pas être piraté, mesurer la mesure dans laquelle il est.

Chapitre 8, 134 à 142

1. Certains des essais de ce livre ont été réécrits, mais à l'exception de la traduction des calculs de probabilité du code Lisp en notation mathématique, j'ai laissé celui-ci tranquille. Donc, certaines choses qui s'y trouvent ne sont plus vraies. Peu de spams contiennent le mot "clic" maintenant. Mais l'algorithme fonctionne toujours. Une version légèrement améliorée attrape environ 99,6 % du spam actuel. Pour en savoir plus sur le filtrage, voir paulgraham.com.
2. En 2002, le taux le plus bas semblait être d'environ 200 \$ pour envoyer un million de spams. C'est très bon marché, 1/50e de cent par spam. Mais filtrer 95 % du spam, par exemple, augmenterait le coût des spameurs pour atteindre un public donné d'un facteur de 20. Peu de gens peuvent avoir des marges assez grandes pour absorber cela.

Chapitre 9, 143-159

1. Sullivan a en fait dit "la forme suit toujours la fonction", mais je pense que la citation erronée habituelle est plus proche de ce que les architectes modernistes voulaient dire.
2. Le moteur du Wright Flyer pesait 152 lb et produisait 12 ch. Le moteur à réaction F414-GE-400 utilisé dans le F-18 pèse 2 445 lb et génère 22 000 lb de poussée. En supposant une poussée de 1 lb = 1 ch, il fournit environ 114 fois plus de puissance par poids.
Les processeurs Intel actuels, quant à eux, offrent environ 1700 fois la puissance de traitement de ceux disponibles il y a 30 ans.

3. Brush, Stephen G., « Pourquoi la relativité a-t-elle été acceptée ? » *La physique en perspective*, 1 (1999), 184-214.

Chapitre 10, 160 à 169

1. La façon la plus courante d'entrer par effraction dans les ordinateurs tire parti de certaines particularités de C. En C, lorsque vous mettez de côté un morceau de mémoire (un *tampon*) pour une entrée que vous attendez, il est alloué à côté de la mémoire contenant *l'adresse de retour* du code en cours d'exécution. L'adresse de retour est la localisation en mémoire du code qui sera exécuté lorsque le code actuel sera terminé. C'est, en fait, la prochaine chose sur la liste des choses à faire de l'ordinateur.

Donc, si quelqu'un veut entrer par effraction dans votre ordinateur, et qu'il suppose que vous utilisez un tampon de 256 octets pour stocker une sorte d'entrée, alors en envoyant un peu plus de 256 octets, il peut écraser l'adresse de retour. Lorsque le code actuel est terminé, le contrôle passera à n'importe quel emplacement en mémoire qu'ils ont spécifié. Et l'emplacement qu'ils spécifieront habituellement sera le début du tampon, qu'ils viennent de remplir avec le programme de langage machine de leur choix. Bingo : leur programme est maintenant en cours d'exécution sur votre ordinateur.

Dans les langages de niveau supérieur, ce serait impossible, mais en C, chaque fois que vous prenez des entrées de l'extérieur et que vous ne vérifiez pas la longueur, vous avez créé une faille de sécurité. Une attaque qui exploite un tel trou est appelée attaque de débordement de tampon. Il existe d'autres moyens d'obtenir le contrôle d'un ordinateur dans une attaque de débordement de tampon, mais l'écrasement de l'adresse de retour est la méthode classique.

Curieusement, les détournements de compagnies aériennes sont également des attaques de débordement de tampon. Dans un avion de ligne ordinaire, les passagers et le poste de pilotage sont adjacents, tout comme les données et le code sont adjacents dans un programme C. En débordant dans le cockpit, les détourneurs passent en effet de l'état de données à celui du code

2. Note aux hackers : ce n'est qu'une métaphore. N'essayez pas de conduire un Yugo avec un moteur à réaction boulonné au toit.

On peut soutenir que le phénomène Yugojet n'est pas nouveau. Fortran doit également sa popularité en grande partie à ses bibliothèques.

3. Cipolla, Carlo, *Guns, Sails, and Empires: Technological Innovation and the Early Phases of European Expansion 1400-1700*, Pantheon, 1965.

Chapitre 11, 170 à 184

1. Je crois que Lisp Machine Lisp a été le premier langage à incarner le principe selon lequel les déclarations (à l'exception de celles des variables dynamiques) n'étaient que des conseils d'optimisation et ne changeraient pas la signification d'un programme correct. Common Lisp semble avoir été le premier à l'indiquer explicitement.

Chapitre 12, 185 à 197

1. Viaweb avait d'abord deux parties : l'éditeur, écrit en Common Lisp, que les gens utilisaient pour construire leurs sites, et le système de commande, écrit en C, qui a dirigé les ordres. La première version était principalement Lisp, car le système de commande était petit.
En janvier 2003, Yahoo a publié une nouvelle version de l'éditeur écrite en C++ et Perl. Mais pour traduire ce programme en C++, ils ont littéralement dû écrire un interpréteur Lisp : les fichiers sources de tous les modèles génératrices de pages sont toujours, pour autant que je sache, du code Lisp. (Voir Greenspun's Tenth Rule p. 216)
2. Robert dit que je n'avais pas besoin d'être secret, parce que même si nos concurrents avaient su que nous utilisions Lisp, ils n'auraient pas compris pourquoi : « S'ils étaient aussi intelligents, ils programmeraient déjà dans Lisp. »
3. Tous les langages sont tout aussi puissants dans le sens d'être équivalents à Turing, mais ce n'est pas le sens du mot dont les programmeurs se soucient. (Personne ne veut programmer une machine de Turing.) Le type de programmeurs de puissance qui se soucient n'est peut-être pas formellement définissable, mais une façon de l'expliquer serait de dire qu'il fait référence à des fonctionnalités que vous ne pourriez obtenir que

dans le langage moins puissant en écrivant un interpréteur pour le langage le plus puissant qu'il contient. Si le langage A a un opérateur pour supprimer les espaces des chaînes et que le langage B n'en a pas, cela ne rend probablement pas A plus puissant, car vous pouvez probablement écrire une sous-routine pour le faire en B. Mais si A prend en charge, disons, la récursivité, et que B ne le fait pas, il est peu probable que ce soit quelque chose que vous pouvez corriger en écrivant des fonctions de bibliothèque.

4. Ou peut-être un treillis, se rétrécissant vers le haut. Ce n'est pas la forme qui compte ici, mais l'idée qu'il y ait au moins un ordre partiel.
5. Il est un peu trompeur de traiter les macros comme une fonctionnalité distincte. En pratique, leur utilité est grandement renforcée par d'autres caractéristiques de Lisp telles que les fermetures lexicales et les paramètres de repos.
6. En conséquence, les comparaisons de langages de programmation prennent soit la forme de guerres religieuses, soit de manuels de premier cycle si résolument neutres qu'ils sont vraiment des œuvres d'anthropologie. Les gens qui apprécient leur paix, ou qui veulent une permanence, évitent le sujet. Mais la question n'est qu'une question à moitié religieuse ; il y a quelque chose qui vaut la peine d'être étudiée, surtout si vous voulez concevoir de nouveaux langages.

Chapitre 13, 198 à 218

1. Après avoir mis cet essai en ligne, j'ai reçu un e-mail apparemment authentique qui commençait :

Cheveux pointues ? Tous les poils ne sont-ils pas pointus ? Si c'est le meilleur terme insultant pour un patron que vous pouvez trouver, il est facile de voir à quel point vous avez mérité le surnom de "nerd".

2. Le processeur IBM 704 avait à peu près la taille d'un réfrigérateur, mais beaucoup plus lourd. Le CPU pesait 3150 livres, et le 4K de RAM était dans une boîte séparée pesant encore 4000 livres. Le Sub-Zero 690, l'un des plus grands réfrigérateurs ménagers, pèse 656 livres.

3. Steve Russell a également écrit le premier jeu informatique (numérique), Spacewar, en 1962.
4. Un certain nombre de fonctionnalités de Lisp, y compris des programmes exprimés sous forme de listes et une forme de récursivité, ont été implémentées dans IPL-V. Mais il s'agissait plus d'un langage d'assemblage ; un programme se composait d'une séquence de paires opcode/adresse. Newell, Allen (éd.), *Information Processing Language-V Manual*, Prentice-Hall, 1961.
5. Si vous voulez tromper un patron aux cheveux pointus pour qu'il vous laisse écrire un logiciel en Lisp, vous pouvez essayer de lui dire que c'est du XML.
6. Muehlbauer,Jen, “Orbitz Reaches New Heights”, *New Architect*, avril 2002.

7 Voici le générateur d'accumulateurs dans d'autres dialectes de Lisp :

```

Scheme: (define (foo n)
            (lambda (i) (set! n (+ n i)) n))
Goo:    (df foo (n) (op incf n _)))
Arc:    (def foo (n) [++ n _])

```

8. Peter Norvig a constaté que 16 des 23 modèles dans les modèles de conception étaient « invisibles ou plus simples » dans Lisp (www.norvig.com/design-patterns).

Chapitre 14, 217 à 235

1. Un programme hello-world est un programme qui ne fait rien d'autre que d'imprimer les mots « Bonjour, monde ! » En Java, vous écrirez :

```

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}

```

Quelqu'un qui n'a jamais écrit de programme regarde probablement cela et se demande, pourquoi avez-vous besoin d'en dire autant pour que l'ordinateur imprime un message ? Curieusement, la réaction des programmeurs expérimentés est identique.

2. Dans *When the Air Hits Your Brain*, le neurochirurgien Frank Vertosick raconte une conversation dans laquelle son résident en chef, Gary, parle de la différence entre les chirurgiens et les internes (« puces ») :

“Gary et moi avons commandé une grande pizza et avons trouvé un stand ouvert. Le chef a allumé une cigarette. « Regardez ces putains de puces, bavardant à propos d'une maladie qu'ils verront une fois dans leur vie. C'est le problème avec les puces, elles n'aiment que les choses bizarres. Ils détestent leurs caisses de pain et de beurre. C'est la différence entre nous et les putains de puces. Vous voyez, nous aimons les grandes hernies discales lombaires juteuses, mais ils détestent l'hypertension. . . . »

Il est difficile de penser qu'une hernie discale lombaire est juteuse (sauf littéralement). Mais je pense que je sais ce qu'ils veulent dire. J'ai souvent eu un insecte juteux à traquer. Certains qui ne sont pas programmeurs auraient du mal à imaginer qu'il pourrait y avoir du plaisir dans un bug. C'est sûrement mieux si tout fonctionne. Et pourtant, il y a indéniablement une sombre satisfaction à traquer certaines sortes de bugs.

Remerciements

La première sur la liste des personnes que je dois remercier est Sarah Harlin. Après avoir écrit un essai, je le lui ai généralement montré en premier. Et elle en a généralement barré la moitié et m'a dit de réécrire le reste. Elle a une oreille parfaite pour le rythme de la prose et aboie à des mots superflus comme un chien après un écureuil.

Si ces essais sont bons, c'est parce que la plupart sont issus de conversations avec elle ou avec Robert Morris, Trevor Blackwell ou Jackie McDonough. J'ai de la chance de les connaître.

Le livre bénéficie des idées de plusieurs autres amis Eric Raymond Je dois des remerciements particuliers non seulement pour ses idées, mais aussi pour son exemple écrit sur le bidouillage informatique.

Je dois remercier beaucoup d'autres pour leur aide et leurs idées, y compris Jülide Aker, Chris Anderson, Jonathan Bachrach, Ingrid Bassett, Jeff Bates, Alan Bawden, Andrew Cohen, Cindy Cohn, Kate Courteau, Maria Daniels, Rich Draves, Jon Erickson, John Foderaro, Bob Frankston, Erann Gat, Phil Greenspun, Ann Gregg, Amy Harmon, Andy Hertzfeld, Jeremy Hylton, Brad Karp, Shriram Krishnamurthi, Fritz Kunze, Joel Lehrer, Henry Leitner, Larry Lessig, Simon London, John McCarthy, Doug McIlroy, Rob Malda, Julie Mallozzi, Matz, Larry Mihalko, Mark Nitzberg, North Shore United, Peter Norvig, the Parmets, Sesha Pratap, Joel Rainey, Jonathan Rees, Guido van Rossum, Barry Shein, the Sloos, Mike Smith, Ryan Stanley, Guy Steele, Sam Steingold, Anton van Straaten, Greg Sullivan, Brad Templeton, Dave Touretzky, Mike Vanier, les We-ickers, JonL White, Stephen Wolfram et Bill Yerazunis.

Ce livre a l'air bien parce que la conception a vraiment été faite par le dieu de la typographie, Gino Lee, pas moi. J'en sais assez sur la conception de livres pour faire tout ce que Gino dit. Chip Coldwell a passé des heures à battre sur les polices et Amy Hendrickson à écrire des macros LaTex pour obtenir l'apparence de la facilité que vous voyez ici. La couverture, curieusement, a été conçue dans un sens par Robert Morris, qui a allumé le Gimp et a fait une intervention chirurgicale sur la version précédente. Merci à Gilberte Houbart

pour son ingéniosité et son insistance à extraire des images de sources du monde entier.

Les gars d'O'Reilly ont fait un excellent travail : Allen Noren, dont l'intérêt général pour la fabrication de bons livres est suffisant pour restaurer sa confiance dans l'industrie du livre ; Betsy Waliszewski, dont la vision d'un livre plus populaire est devenue furtivement la mienne ; Matt Hutchinson, Robert Romano et Claire Cloutier, qui ont fait en sorte que la production se déroule sans heurts ; et Tim O'Reilly, qui montre ce que l'édition peut être lorsqu'un éditeur est une personne plutôt qu'un conglomérat.

Un grand merci à Jessica Livingston. Ses conseils ont amélioré une partie de ce livre, de la couverture à l'index. Son encouragement indéfectible a également amélioré le livre : en me disant constamment que beaucoup de gens voudraient le lire, elle m'a fait peur de m'efforcer d'en faire quelque chose que beaucoup de gens voudraient lire.

J'ai appris le bidouillage de beaucoup de gens, mais j'ai appris à peindre principalement d'un : Idelle Weber, une grande enseignante, d'autant mieux pour enseigner par l'exemple. Je suis profondément redevable à elle et à son mari Julian pour des années de gentillesse.

Merci enfin à mon père, pour m'avoir enseigné le scepticisme, et à ma mère, pour m'avoir enseigné l'imagination. L'avoir comme mère a été comme voir le monde en couleur.

Crédits des images

32. Léonard De Vinci, *Ginevra de' Benci*, Ailsa Mellon Bruce Fund. Droit d'auteur de 2004. Conseil d'administration, National Gallery of Arts, Washington.
36. Copyright Archivo Iconografico S.A. /Corbis.
52. Photographie par Margaret Wozniak. Reproduit par l'autorisation de Steve Wozniak.
93. Copyright *Popular Electronics*. Courtoisie du Computer History Museum.
95. Courtoisie du service de police d'Albuquerque.
148. Photographiée et reproduite avec la permission de John Colley.
152. Photographie d'Alexei Nabarro, via iStockphoto.
153. La Collection Royale. Droit d'auteur de l'image 2004, Sa Majesté la reine Elizabeth II.
155. Courtoisie de la NASA Dryden Photo Collection.
156. Musée d'histoire de l'art, Vienne.
200. Reproduit avec la permission de Lawrence Livermore National Labs.
201. Reproduit avec la permission de John McCarthy.

Glossaire

Abstrait : cacher les détails. Lorsqu'un langage est plus abstrait, vous pouvez écrire des programmes en utilisant un plus petit nombre d'opérations (individuellement plus puissantes).

Arbre : Structure de données dont chaque instance peut renvoyer à deux ou plusieurs autres instances. Par exemple, un arbre généalogique.

Arbre d'analyse : La *structure de données* dans laquelle un compilateur qui compose votre programme, comme première étape de sa traduction en *langage machine*.

Ada : Un langage *orienté objet* conçu par un comité du DoD à la fin des années 1970. Il s'est avéré à peu près comme vous vous y attendiez.

Administrateur système : Quelqu'un qui installe du matériel et des logiciels informatiques, et maintient les réseaux en bon fonctionnement.

Algol : Un langage de programmation initialement conçu en 1958 par un comité (mauvais) de personnes très intelligentes (bonnes). Rarement utilisé pour écrire des programmes, mais a eu une grande influence sur les langues suivantes.

Algorithme : une méthode pour faire quelque chose. Les recettes sont des exemples d'algorithmes.

API : (Application Program Interface) : Interface de Programme d'Application. La liste des commandes qu'un système d'exploitation ou une *bibliothèque* acceptera des *applications*.

APL : Un langage extrêmement succinct conçu au début des années 1960 par Ken Iverson. Utilisé surtout dans les applications numériques. Son descendant moderne est J.

Application : Un programme qui n'est pas une infrastructure. Par exemple, un traitement de texte, mais pas un *système d'exploitation*. Pas un terme précis.

Arc : Un dialecte *vaporware* de Lisp.

ASP (*Application Service Provider*) : Fournisseur de Service d'Applications. Une entreprise qui vous permet d'utiliser des logiciels sur leurs ordinateurs via un réseau, au lieu d'installer et d'exécuter le logiciel sur votre propre ordinateur.

B&D Language : Langage de bondage et de discipline. Un langage qui fait en sorte que le programme suive des règles strictes.

Bandé passante : La vitesse à laquelle une connexion peut transmettre des données.

Bayésien : Utilisation de la règle de Bayes, qui indique comment combiner les preuves statistiques

Bibliothèque : Une collection de codes existant pour effectuer une tâche spécifique.

Binaire : Lorsqu'il est utilisé avec un article (par exemple « un binaire »), *code objet*. Lorsqu'il est utilisé sans un article, une façon de représenter les nombres en base 2 au lieu de la base 10 plus familière. Les chiffres successifs (en commençant par la droite) représentent des puissances de deux. Ainsi, 101, en binaire, représente le nombre que nous écrivons comme 5 en décimal. La plupart des ordinateurs représentent les données en binaire, parce qu'il est plus facile de concevoir des circuits à deux états (on ou off) qu'à dix.

Bloatcode : Un programmeur qui rend les programmes plus longs qu'ils ne devraient l'être.

Blub Paradox : L'incapacité à comprendre la puissance des langages de programmation plus puissants que ceux auxquels vous avez l'habitude de penser.

Boucle infinie : Voir *définition circulaire*.

Branche : Une commande *go-to* en langage machine.

Bug : Une erreur dans un programme. Avant les ordinateurs ; au début du XXe siècle, il était courant de parler de "repasser les bugs" dans une pièce de théâtre de Broadway.

C : Un langage magnifiquement simple développé par Dennis Ritchie au début des années 1970. Largement utilisé dans les infrastructures telles que les *systèmes d'exploitation* et les routeurs.

C++ : Une tentative d'ajouter des *capacités orientées objet* au langage C, conçue par Bjarne Stroustrup en 1983. Populaire parce que sa syntaxe est similaire à celle de C, et qu'elle peut être mélangée avec des programmes C.

Calcul des nombres : Effectuer des opérations simples sur de grandes quantités de données numériques.

Chaîne : Une séquence de caractères

Caractères alphanumériques : Lettres et chiffres

Champ : L'une des parties d'une *structure de données*.

Checksum : Une façon de résumer toutes les informations d'un fichier pour obtenir un nombre qui peut être utilisé pour l'identifier. Une (pas très bonne)

Classe : En programmation *orientée objet*, un type de données.

Click trail : Série de requêtes HTTP envoyées à un *serveur web* par un utilisateur spécifique. Cela correspond généralement à la série de pages web qu'ils ont visitées.

Client : Un ordinateur ou un appareil qui soumet des demandes à un serveur.

Conception prématuée : Décider trop tôt de ce qu'un programme doit faire.

Cobol : Un langage primitif conçu au début des années 1960 pour être utilisé dans les applications commerciales. Il n'a été remplacé que récemment par Java en tant que langage le plus populaire.

Code : Lorsqu'il n'est pas qualifié, *code source*.

Code byte : Tout langage semblable au *langage machine*, mais pas celui d'un ordinateur spécifique. Étant semblable au *langage machine*, il est facile d'écrire un *interprète* de code byte, qui lit les programmes de code byte et exécute les commandes correspondantes en langage machine.

Code objet : *Langage machine*, résultat d'un *compilateur*.

Coincé : Dans un état de non-réponse. Dit surtout d'un serveur.

Collecte des ordures : Récupération automatique de la mémoire dont un programme n'a plus besoin, au lieu de demander au programmeur de déclarer explicitement (et souvent à tort) quand il a fini de l'utiliser.

Colocalisé : Situé en particulier chez des fournisseurs d'accès à internet

Commentaire : Partie d'un programme qui est ignorée par l'ordinateur. Généralement insérée en tant qu'annotation pour les lecteurs humains

Common Lisp : Un dialecte populaire de Lisp conçu par un comité dans les années 1980.

Compilateur : Un programme qui traduit des programmes écrits dans un langage plus puissant et plus succinct (*un langage de haut niveau*) en des commandes plus simples (*langage machine*) que le matériel informatique comprend. Voir aussi : *interprète*.

Complexité : La complexité temporelle d'un algorithme est la vitesse à laquelle le temps nécessaire à sa réalisation croît lorsque la taille de l'entrée augmente. Par exemple, si vous devez rechercher une chambre pour une personne spécifique en regardant chacune d'elles à tour de rôle, le temps nécessaire pour la trouver sera proportionnel au nombre de personnes. Un tel algorithme est appelé $O(n)$, ce qui signifie qu'il prend du temps proportionnel à n , la taille des données. Alors que si vous vouliez trouver les deux personnes dans la pièce qui ressemblaient le plus à des frères et sœurs, vous prendriez probablement du

temps proportionnel au carré du nombre de personnes, parce que vous pourriez avoir à comparer chaque paire, et le nombre de paires est le carré du nombre de personnes. Un tel algorithme est $O(n^2)$.

Conditionnel : Une *expression de langage de haut niveau* (ou une *instruction*) dans laquelle un code différent est exécuté selon qu'une condition est vraie ou non. Par exemple : s'il fait beau, alors allez vous promener, sinon restez à l'intérieur et lisez.

CPU (Central Processing Unit) : Unité centrale de traitement. Partie d'un ordinateur, généralement constituée d'une seule puce, où les calculs sont effectués. Le concept devient de plus en plus flou, parce qu'il y a maintenant des processeurs dans les cartes graphiques et les disques durs.

Crash : Lorsqu'un bug provoque l'arrêt du fonctionnement d'un système d'exploitation ou d'une application. Ou encore, dans le cas des disques durs, un dysfonctionnement matériel.

Cruft : Débris.

Cycle : Le temps minimum nécessaire à l'exécution d'une instruction de machine. Un ordinateur avec une vitesse d'horloge de 1 GHz a un milliard de cycles par seconde, ce qui signifie qu'il peut exécuter jusqu'à un milliard d'instructions de machine par seconde.

Dactylographie statique : Un langage est tapé systématiquement si le type de valeur que chaque variable peut avoir doit être connu.

DARPA (Defense Advanced Research Projects Agency) : Agence des projets de recherche avancée de défense. A financé une grande partie de la recherche en informatique aux États-Unis.

Dactylographie dynamique : Le contraire de la *dactylographie statique*.

Débogage : Trouver et corriger les erreurs dans un programme.

Déclaration : Un quantum de code qui ne donne pas de valeur. Pour être utilisé, il doit donc avoir un certain effet, par exemple : imprimer quelque chose. On peut soutenir que tout concept est une erreur; dans certains langages, il n'y a que des expressions

Définition circulaire : voir *boucle infinie*.

Déprécié : Dit des pratiques soutenues par une norme dont les auteurs regrettent aujourd'hui qu'elles n'aient pas été autorisées.

Dialecte open source d'Unix : Appelé GNU Linux par les fastidieux, car alors que le noyau (la partie la plus interne) a été écrit par Linus Torvalds, une plus grande partie du code provient du projet GNU de Richard Stallman.

Diff : Une comparaison non sélective et microscopiquement complète entre deux versions de quelque chose. À partir de l'utilitaire Unix diff, qui compare les *fichiers*.

Environnement : Logiciel d'aide à l'écriture de programmes, par exemple éditeurs et *profileurs*.

En retrait : Comme un plan...

En-tête : La partie en haut d'un email contenant des informations à ce sujet. L'utilisateur ne voit que les lignes DE, A, DATE, OBJET et CC, mais il y en a d'autres qui décrivent par exemple le chemin emprunté par l'email.

E/S (Entrée/Sortie) : I/O input/output : Généralement, impression et lecture de caractères de données binaires

Expression : Un quantum de code qui, lorsqu'il est exécuté, donne une valeur. Par exemple, l'expression $2 + 3$ donnera 5.

Expression régulière : Motif utilisé comme un tamis pour récupérer des éléments aux *chaînes de caractères*.

Fermeture lexicale : Une *fonction* qui fait référence à une variable définie non pas à l'intérieur, mais dans le code environnant.

Fichier : Séquence de caractères ou de chiffres *binaires*, généralement stockée sur un disque.

Filtrage basé sur le contenu : Filtrage des courriers électroniques en fonction de leur contenu et non, par exemple, de leur provenance sur internet.

Fonction : Sous-programme qui, lorsqu'il est appelé, produit une valeur qui devient la valeur de l'appel. Dans certains langages, les fonctions sont un *type de données*.

Fortran : Un langage de programmation largement utilisé pour les applications numériques. Conçu à l'origine par un groupe d'IBM en 1956, il a beaucoup évolué depuis.

FreeBSD : Dialecte *open source* d'*Unix*

Freeware : Logiciel distribué gratuitement.

Guerre du design : Un concours où le meilleur design l'emporte, plutôt que, par exemple, le marketing ou le contrôle des points de vente

Glue program : Un programme pour séquencer ou déplacer des données entre les *applications*.

Goto : Une commande qui transfère le contrôle à une autre partie d'un programme. Parce qu'il n'y a pas de mécanisme pour revenir à un goto, comme il y a un appel de sous-routine, les programmes qui utilisent des gotos ont tendance à devenir des *spaghettis*. Rare maintenant.

Greenspun's Tenth Rule : Tout programme *C* ou *Fortran* suffisamment compliqué contient une implémentation lente, ad hoc et informellement spécifiée, criblée de bogues, de la moitié de *Common Lisp*

Hack : Une solution qui enfreint d'une manière ou d'une autre les règles. Peut être bon ou mauvais.

Hacker (1) Un bon programmeur. (2) Quelqu'un qui s'infiltre dans les ordinateurs.

Heuristique : Règle empirique.

High level : Substantiellement plus abstrait que le langage de la machine.

HTML (HyperText Markup Language) : La notation utilisée pour exprimer les pages Web.

HTTP (HyperText Transfer Protocol) Le protocole que les navigateurs Web et les serveurs utilisent pour communiquer entre eux.

Hypothèse de Brooks : Le nombre de lignes de code que les programmeurs peuvent produire par jour est constant, quelque soit le langage utilisé.

IA (Intelligence artificielle) : Terme général pour plusieurs types de travaux qui tentent de faire réfléchir les machines

Ingénieur logiciel : Terme formel pour programmeur.

Interprète : Comme un compilateur, un interprète accepte les programmes écrits dans un langage de haut niveau, mais au lieu de traduire l'ensemble du programme en langage machine, puis de l'exécuter, l'interprète examine le programme une pièce à la fois et exécute les commandes correspondantes en langage machine.

Inner loop : Partie d'un programme qui a été exécuté particulièrement souvent.

Instrument : Pour modifier un programme pour garder une trace de tout ce qu'il fait, de sorte que s'il est lent ou utilise trop de mémoire, vous pouvez savoir pourquoi.

Instruction de machine : Une commande de langage de machine.

Intel box : Un ordinateur avec un processeur Intel.

Java : Une tentative d'un meilleur C++ par James Gosling. Initialement appelé Oak, il a été renommé Java par Sun lorsqu'ils l'ont adopté dans l'espoir d'insérer une couche contrôlée par Sun entre les systèmes d'exploitation et les applications. Cela n'a pas fonctionné, mais Java est populaire de toute façon, en partie en raison de l'énorme effort de marketing de Sun, et en partie parce qu'il y a une demande pour un meilleur C++.

Javascript : Un langage de script pour les navigateurs Web conçu par Brendan Eich. Il n'a pas de connexion intrinsèque à Java, qui est dans la plupart des cas inférieur. Indûment calomnié parce qu'il est principalement utilisé pour faire des choses ringardes sur des sites web.

Jetons : Séquence de caractères formant une unité. Terme plus général pour "mot".

Kludge : Un mauvais *hack*. (Rime avec *stooge*.)

Langage d'assemblage : Forme plus conviviale pour les programmeurs que le *langage machine*. Les commandes sont les mêmes, mais vous pouvez utiliser des noms plus pratiques.

Langage de programmation : Un langage de haut niveau et ce que le compilateur utilise comme entrée pour générer du code objet (Je plaisante, voir Chapitre 10).

Langage de script : Un langage utilisé pour personnaliser un programme. Parfois les langages open source comme Perl et Python sont appelés langages de scripts, mais cet usage n'a pas de sens.

Langage intégré : Un langage défini à l'intérieur d'un autre langage, généralement pour un type de problèmes spécifiques. Par exemple, si vous définissez une série de commandes pour manipuler des images, vous pouvez commencer à les considérer comme un langage de manipulation d'images. Voir *Programmation ascendante*.

Langage machine : La liste des commandes qu'un processeur sait obéir. Aussi, une séquence de telles commandes.

Langage pour les personnes intelligentes : Un langage qui met le pouvoir sur la sécurité. Une collection de code existant pour effectuer une tâche spécifique.

Lié : Contraint par une ressource particulière. Par exemple lié par l'E/S, lié par la mémoire, lié à l'unité centrale.

LFSP : *Language For Smart People*; un langage qui privilégie le pouvoir de sécurité

Linux : Un dialecte *open source* d'*Unix*. Appelé GNU Linux par les plus pointilleux, car le noyau (la partie la plus interne) a été écrit par Linus Torvalds, et la plus grande partie du code provient du projet GNU de Richard Stallman.

Lisp : Une famille de langues dérivée d'un langage par John McCarthy à la fin des années 1950. Les deux dialectes les plus connus sont *Common Lisp* et *Scheme*. Les langages *open source* récents contiennent des quantités croissantes d'ADN Lisp.

Liste : Une série de données, souvent de *types* différents, qui peuvent être réunies comme des trains pour faire des listes plus grandes.

Logiciel hérité : Logiciel dont une organisation a encore besoin, qui n'est pas écrit comme ils le souhaiteraient, et qu'elle ne peut pas se permettre ou n'ose pas réécrire.

Loi de Moore : La version officielle de la Loi de Moore est que le nombre de transistors sur une puce double tous les deux ans. Mais la plupart des gens utilisent ce terme pour signifier que les processeurs sont deux fois plus rapides tous les 18 mois. Sans doute plus de plan d'affaires que de droit, parce que Gordon Moore a été l'un des fondateurs d'Intel.

Loi de Parkinson : Que les ressources nécessaires pour accomplir une tâche seront étendues pour consommer les ressources nécessaires.

Low-level : Moins *abstrait* ; n'autorisant que des commandes simples, comme le *langage machine*.

Machine d'état : Une machine théorique qui peut être dans un certain ensemble d'états possibles, avec des connexions entre les états lorsque certaines conditions sont vraies.

Machine de Turing : Ordinateur imaginaire simple dont les propriétés sont utilisées pour prouver des théorèmes sur l'informatique. On pense actuellement qu'il est impossible d'obtenir quelque chose de plus puissant, dans le sens où l'on ne peut pas définir un ordinateur dont les programmes ne pourraient pas être traduits en programmes de machines de Turing. Mais personne ne peut le dire avec certitude, car le terme "ordinateur" n'est pas formellement défini.

Macro : Un programme qui génère des programmes. Les moyens de le faire varient entre les langues, de sorte qu'un "macro" dans une langue peut signifier quelque chose de beaucoup plus puissant que dans une autre.

Mainframe : Un gros ordinateur basé sur des conceptions des années 1960 et 1970.

Maladie de l'administrateur système : La croyance implicite des administrateurs systèmes selon laquelle l'infrastructure qu'ils supervisent est une fin en soi, plutôt qu'un outil pour les utilisateurs. Plus généralement, l'attitude selon laquelle les clients sont une nuisance, plutôt que la raison pour laquelle votre travail existe. Endémique dans les emplois non exposés à la concurrence.

Manipulation de bits : effectuer des transformations simples sur de grandes zones de la mémoire d'un ordinateur. Par ex. déplacer une fenêtre sur l'écran

Math envy : L'inquiétude de ne pas être aussi intelligent que les mathématiciens, surtout lorsqu'elle se manifeste dans un travail à saveur gratuitement mathématique.

Méta circulaire : Lorsque *l'interprète de la langue* est écrit dans cette langue. Plus une technique pour décrire les langues que pour les implémenter.

Méthode : Dans la programmation *orientée objet*, un sous-programme considéré comme une propriété d'une *classe* de choses. Par exemple, la méthode d'aire de la classe de cercle pourrait être une sous-routine pour calculer les aires des cercles.

Module : Un groupe de sous-programmes et de variables considérés comme une unité. En général, seuls ceux spécifiquement notés sont accessibles au code en dehors du module.

Objet : Un terme aux significations multiples. Dans le sens le plus général, une instance d'un *type de données*. Par exemple, une *chaîne* particulière ou un entier particulier.

OO, orienté-objet : une façon d'organiser les programmes de manière à ce que le code permettant d'effectuer une certaine tâche sur différentes classes de données soit divisé en éléments distincts (méthodes) pour chacune d'entre elles.

Occam's Razor : Qu'il faut préférer la plus simple des deux théories.

Opensource : Dont le *code source* est librement distribué et peut être modifié par n'importe qui, généralement à condition que les modifications soient également mises à disposition gratuitement. *Linux* et *FreeBSD* sont des *systèmes d'exploitation* open source bien connus.

Orthogonal : Indépendants les uns des autres et donc combinables de nombreuses manières. Les Lego classiques sont plus orthogonaux que les maquettes en plastique.

Optimisation : Changer un programme pour le rendre plus efficace.

Optimisation prématurée : Réglage d'un programme pour les performances avant de l'écrire. L'équivalent logiciel de se marier jeune.

Ordinateur parallèle : Un ordinateur dont le matériel peut effectuer plusieurs calculs simultanément. Ce n'est pas une catégorie clairement délimitée, car tous les processeurs modernes utilisent un certain nombre de parallélismes pour augmenter la vitesse.

Parser : Un programme qui lit l'entrée et produit un *arbre d'analyse*.

Pascal : Dérivé d'Algol conçu au début des années 1970 par Niklaus Wirth.

Patch : Un morceau de code publié pour corriger une faille dans un programme précédent.

PDA, Personal Digital Assistant (Assistant numérique personnel) : Un petit ordinateur que vous emportez avec vous. A généralement une interface plus facile mais plus limitée qu'un ordinateur ordinaire.

Perl : Un *langage open source* développé par Larry Wall. Initialement destiné à manipuler des chaînes de caractères, il est devenu populaire parce que c'est une grande partie de ce que font les programmeurs. Célèbre pour sa *syntaxe* complexe (mais concise) et son évolution rapide et promiscue.

Pipe : Une façon de joindre les commandes du système d'exploitation afin que la sortie de l'une devienne l'entrée d'une autre.

Pointeur : Un morceau de données dont la valeur est l'emplacement dans la mémoire d'un autre.

Pointeur arithmétique : Trouver des éléments dans la mémoire en ajoutant certaines quantités à des emplacements déjà connus. Technique de *bas niveau*.

Patron aux cheveux pointus : Personnage de la bande dessinée *Dilbert* de Scott Adams. Généralement, un cadre moyen inepte et autoritaire.

Pilote de périphérique : Composant d'un *système d'exploitation* qui sait comment communiquer avec un périphérique spécifique, comme une imprimante.

Polynôme : Appliqué à la croissance, signifie que y croît comme une puissance de x, par exemple comme le carré ou le cube de x. La courbe qui en résulte devient plus raide avec le temps.

Portable : Capable d'être déplacé vers du nouveau matériel. Les programmes écrits en *langages de haut niveau* sont (plus) portables que les programmes en *langage machine*, car ils ne font (presque) rien sur le matériel.

Programmation ascendante : Un style de programmation qui fonctionne à partir de l'autre direction que le style descendant précédent. Au lieu de subdiviser une tâche en unités plus petites, vous construisez un "langage" d'idées vers votre tâche. Les deux techniques peuvent être combinées.

Portail : Site Web.

Processus : Dans un *système d'exploitation* qui peut contrôler plusieurs programmes à la fois (comme tous les systèmes d'exploitation modernes), l'un de ces programmes.

Programme jetable : Un programme écrit pour satisfaire un besoin temporaire.

Profiler : Un programme qui surveille votre programme pendant qu'il est en cours d'exécution et vous indique quelles parties consomment le plus de ressources. Voir la *boucle intérieure*.

Pseudocode : Un langage pour exprimer des *algorithmes* "sur papier" plutôt que pour les ordinateurs. On peut soutenir que tout ce concept est un artefact d'utilisation de langages de trop bas niveau.

Python : Langage *open source* développé par Guido van Rossum. Fortement *orienté-objet*, il est considéré par ses fans comme une alternative plus propre à *Perl*.

QA (Quality Assurance) : Assurance qualité. Dans les logiciels, les personnes qui détectent et cataloguent les *bugs*.

RAID (Redundant Array of Independent Disks) Réseau redondant de disques indépendants. Un morceau de matériel qui utilise plusieurs disques durs pour simuler un disque dur qui (en théorie) ne plante jamais.

Read-eval-print loop : Un *toplevel*.

Récursif : Un *algorithme* qui se réfère à lui-même. L'algorithme d'un policier pour interroger les gens est récursif : demandez à la personne si elle est au courant du crime, ou si elle connaît quelqu'un qui le sait, et si c'est le cas, interrogez-la aussi.

Représentation littérale : Une façon de se référer directement dans un *langage de haut niveau*. Dans la plupart des langages la représentation de 5 est 5 (l'expression 2+3 a la même valeur, mais ce n'est pas une représentation littérale).

RISC (Reduced instruction Set Computer) : Ordinateur de jeu d'instructions réduit . Un ordinateur dont les commandes en *langage machine* font peu, mais fonctionnent vite. L'objectif est de faire une meilleure cible pour les *compilateurs*, de la même manière que le film de granularité fine donne des images plus nettes.

Ruby : Un nouveau concurrent open source pour *Perl* et *Python* développé par Yukihiro "matz" Matsumoto.

Scan : Pour regarder une série de caractères et les diviser en *jetons*.

Schéma : Un dialecte élégant mais primitif de *Lisp* conçu par Guy Steele et Gerry Sussman en 1975.

Script CGI (Common Gateway Interface) : Script d'interface de passerelle commune. Programme qu'un serveur web exécute lorsqu'il doit calculer quelque chose (par exemple les résultats d'une recherche) au lieu de vous envoyer vers une page web préexistante. La principale limitation des scripts CGI est qu'ils ne génèrent qu'une seule page avant de se terminer, au lieu de rester en mémoire et d'avoir une conversation continue avec l'utilisateur comme les langages de bureau

Serveur : Un ordinateur sur un réseau qui répond aux demandes d'autres ordinateurs.

Serveur web : Un serveur qui répond aux demandes HTTP.

SETI@home (Search Extra-Terrestrial Intelligence etc.) : Un projet de recherche de signaux d'autres formes de vie dans l'électromagnétique à la recherche de signaux d'autres formes de vie, en utilisant les cycles d'ordinateurs de bureau connectés à l'internet.

S-expression : Un *jeton*, ou zéro ou plusieurs s-expressions entre parenthèses.

Smalltalk : Le langage orienté objet canonique, conçu par Alan Kay en 1972.

SO, système d'exploitation : Programme qui contrôle l'exécution d'autres programmes (*Unix*, *FreeBSD*, *Linux*, *OSX* et la famille *Windows*)

Socket : Dans *Unix*, un canal par lequel les processus peuvent communiquer à travers un réseau.

Spaghetti : Code dont la structure a tellement de rebondissements que personne ne peut le comprendre, y compris l'auteur.

Spam : E-mail de masse non sollicité, généralement de la publicité. D'un sketch de Monty Python dans lequel les Vikings étouffent la conversation avec des refrains de "Spam, Spam, Spam".

Spec : Spécification. Une description informelle de ce qu'un programme devrait faire.

SSH (Secure SHELL) : Un programme pour se connecter en toute sécurité à un ordinateur distant.

SSL (Secure Sockets Layer) : Un protocole pour la transmission sécurisée des données sur le Web.

Sous-routine : Un morceau de code distinct. Lorsque, à un moment donné dans un programme, vous voulez exécuter ce code, vous lappelez, et lorsque la sous-routine est terminée, le contrôle revient au point où l'appel a eu lieu. Dans un livre de cuisine, une recette pour faire du glaçage pourrait être une sous-routine d'une recette de gâteau, et l'appel pourrait être "faire du glaçage en utilisant la recette de la page x".

Sous-ensemble : Un concept inclus dans un autre. La cuisson est un sous-ensemble de la cuisine.

Suits : les personnes non techniques, en particulier les cadres. Dérive des vêtements qu'ils portaient avant de s'habiller comme des hackers dans les années 1990.

Structure de données : Format pour les données comportant plusieurs parties. Par exemple, vous pouvez utiliser un composé d'une paire de nombres pour représenter des points sur un graphique.

Structure en blocs : Décrit un langage dans lequel les programmes ont des parties subsidiaires au lieu d'être une simple liste de commandes.

Start-up larvaire : Une start-up dans la phase la plus précoce, lorsque les fondateurs ne sont pas sûrs de vouloir créer une entreprise.

Symbol : Un *type de données* dont les instances sont des *jetons*. Comme les *chaînes*, sauf (a) un symbole est une seule unité, pas une séquence de caractères, et (b) il n'y a généralement qu'un seul symbole avec un nom donné, alors qu'il peut y avoir plusieurs chaînes contenant les mêmes caractères.

Syntaxe : La forme utilisée pour exprimer les idées dans un programme. Pour donner à x la valeur 10, différentes langues pourraient dire $x = 10$, $x <- 10$, ou $(= x 10)$.

Tableau : Ce que l'on appelait à l'école une matrice : une collection à n dimensions de casiers numérotés pour stocker des données.

Table de hachage : Une structure de données comme une base de données dans laquelle vous pouvez stocker des morceaux de données stockées sous des clés individuelles et récupérer plus tard les données stockées sous une clé donnée

Tampon : Segment de mémoire utilisé pour contenir une séquence de données que le programme attend en entrée ou qu'il accumule en vue de la sortie.

Technologie d'information : L'infrastructure informatique, ou les personnes chargées de la maintenir. Terme utilisé principalement dans les grandes entreprises ou les entreprises non-techniques.

Toplevel : Une interface avec un langage de programmation dans laquelle vous avez une conversation continue avec le langage, comme vous le faites avec Unix, plutôt que de simplement compiler des programmes et de les exécuter.

Turing-complete : Un langage est Turing-complet si tout programme écrit dans ce langage peut être traduit en un programme de machine de Turing et vice versa. Tous les langages de programmation sont Turing-complets, ce qui signifie qu'ils sont tous (d'un point de vue théorique) équivalents en puissance.

Type : *Type de données.*

Type de données : Catégories de données qu'un langage peut traiter. Les types de données typiques sont les entiers (1), les nombres à virgule flottante (1,234) et les chaînes de caractères “monstres”

UDP : Protocole de diffusion d'informations sur les réseaux.

UI (User interface) : Interface utilisateur

Unix : Le *système d'exploitation* à partir duquel les plus actuels dérivent. Le terme est utilisé à la fois de manière générique et est une marque d'une entreprise qui s'est terminée avec les droits d'une variante précoce. Développé à l'origine à Bell Labs par Ken Thompson et Dennis Ritchie au début des années 1970.

Uptime : Pourcentage de temps pendant lequel un ordinateur, en particulier un *serveur*, fait ce qu'il est censé faire. Il s'agit également du temps écoulé depuis la dernière panne d'un ordinateur.

URL (*Uniform Resource Locator*) : L'adresse d'une page web. Plus précisément, une demande à un *serveur web*, généralement pour une page web, mais peut-être pour exécuter un programme (e.g. une recherche web).

Utilisateur final : Euphémisme pour utilisateur non averti.

Vaporware : Un logiciel dont on parle mais qui n'est pas encore disponible.

VC (*venture capitalist*) : Personne qui fournit de l'argent pour créer ou refinancer une entreprise, en échange d'une partie des actions.

Vecteur : Tableau à une dimension ; une séquence.

Version 1.0 : La toute première version de quelque chose, avec l'implication qu'elle sera incomplète ou brisée.

VT100 : Un terminal informatique populaire dans les années 1980.

Wysiwyg (“What you see is what you get”) : "Ce que vous voyez est ce que vous obtenez." (Pronounced whizzy wig.) E.g. un processeur de texte où ce que vous voyez à l'écran ressemble à la page qui sortira de votre imprimante.

XML : Format d'organisation des données.

