LES AMIS DE
l'École de Paris

http://www.ecole.org

LOGICIELS : COMMENT CHASSER LES BUGS ?

par

Gérard BERRY
Directeur de Centre de Mathématiques Appliquées
de l'École des mines de Paris

Séance du 4 octobre 1996
Compte rendu rédigé par Pascal Lefebvre

En bref

S'il vous fallait préparer à l'avance tous les ordres à donner à des individus totalement stupides, mais absolument obéissants, pour qu'ils réalisent une tâche complexe, vous diriez que c'est là un problème de management nouveau : les gens ne sont, en général, ni stupides, ni obéissants, ce qui aide à faire face à l'imprévu. Si, en plus, la moindre erreur provoque une catastrophe, vous diriez que la réussite d'un tel plan d'action tient du miracle. Vous venez pourtant de définir ce qu'est la fabrication d'un logiciel. Pour que la réussite ne tienne pas que du miracle, il faut donc une rigueur de fer, pas mal de culture, et de bons outils tels que ceux qui apparaissent aujourd'hui, ce qui n'est malheureusement pas encore assez connu.

EXPOSÉ de Gérard BERRY

Dans l'informatique, il faut considérer deux choses, encore assez disjointes : le matériel et le logiciel. Le matériel marche étonnamment bien, est extrêmement fiable et ne coûte plus rien. Il y a pourtant là un premier point culturel intéressant : sa performance double tous les dix-huit mois depuis quinze ans et ça sera comme ça pendant longtemps. C'est un fait connu, et pourtant, les gens en sont encore surpris, ce qui est surprenant.

Le logiciel, par contre, marche beaucoup moins bien. Dans les entreprises, il y a de grands échecs et dans une proportion qu'on ne retrouve pas pour d'autres technologies, alors que l'informatique a déjà cinquante ans. Nous allons voir pourquoi. Notons d'abord que le logiciel est d'une variété immense. Je distingue trois sortes de systèmes informatiques : l'informatique classique, l'informatique interactive et l'informatique réactive.

Jusque dans les années 1970, l'informatique se contente de collecter des données et de délivrer des résultats : c'est l'informatique de gestion, la CAO, le calcul scientifique, etc. Techniquement, ces choses sont assez bien comprises et ce n'est pas là qu'on a les plus grandes difficultés. On se dirige aujourd'hui vers une informatique nouvelle, au sein de laquelle on peut distinguer deux formes.

La première, l'informatique interactive, est caractérisée par l'interaction hommes-informatique : on met machines et hommes en réseau. C'est le cas des banques, des systèmes de réservation, d'Internet, etc. Ces systèmes sont plus complexes et plus compliqués à mettre au point que les systèmes classiques. Les possibilités d'erreurs sont beaucoup plus nombreuses et leurs conséquences plus lourdes. Dans ces systèmes, c'est la machine qui est le maître : on attend qu'elle veuille bien répondre et c'est elle qui décide quand les choses se font. Longtemps, des mythes ont imprégné ce domaine, du style : *« Pour rendre l'informatique plus facile, il suffit de la répartir »*. Ceux qui ont essayé savent pourtant à quel point il n'est pas simple de répartir l'informatique.

La deuxième est constituée par les systèmes réactifs, caractérisés par l'interaction machine-environnement physique. Là, c'est l'environnement qui est le maître : quand on pilote un Airbus - objet très informatisé - ce n'est pas à l'ordinateur de choisir le rythme de l'interaction. Ce domaine pose des problèmes techniques assez différents, mais surtout les erreurs y sont catastrophiques. On y trouve des gros systèmes - avions, trains, autos -, mais aussi de tout petits, comme les appareils photos, et une bonne partie des télécommunications.

Dans chacun de ces trois types de systèmes, les choses sont différentes : une erreur dans Word, c'est ennuyeux ; une erreur dans Ariane, c'est cinq milliards !

Sur le plan technique, l'informatique est assez insensible aux applications et on peut travailler dans les domaines les plus divers avec le même outillage. C'est un peu comme les équations différentielles : on les utilise en mécanique, en chimie ou ailleurs. C'est donc une science relativement autonome.

Logiciels et progiciels

Au début, il n'y avait que des logiciels à façon. Puis sont arrivés les progiciels. Maintenant, il y a deux mondes assez disjoints, fort bien décrits par G. Dréan[1].

Les progiciels sont des logiciels qu'on peut acheter en grande surface, sur étagère comme on dit en anglais. On les trouve dans le calcul scientifique, le traitement d'image, la bureautique, etc. C'est un très gros marché et Microsoft y est l'acteur prépondérant. Le marché des logiciels à façon, tirés à petit nombre d'exemplaires, reste cependant encore très important. Il y a beaucoup de logiciels dans un Airbus, mais pas beaucoup de clients en dehors.

[1] G. Dréan, *L'industrie informatique, structure, économie, perspectives*, Masson, 1996. Voir aussi : *Les structures de l'industrie informatique et les comportements des entreprises*, École de Paris, mars 1996.

On a toujours besoin de produits standardisés et de produits spécifiques, ce n'est pas particulier à l'informatique. Tout le monde s'efforce de travailler davantage par assemblage de composants préfabriqués mais ce n'est pas si simple : la nature résiste. C'est certainement une des voies d'avenir, mais, en attendant, on garde souvent une façon assez primaire de travailler.

Un monument invisible

Un logiciel est un gros objet invisible. On peut l'imprimer sur un listing, mais ça ne sert pas à grand-chose car le lire et le comprendre est très ardu. De plus, c'est très procédurier. Essayez donc de décrire la marche d'une entreprise dans tous ses détails, en posant comme principe que vos interlocuteurs sont intégralement stupides et obéissants : vous allez écrire des millions d'instructions mais personne n'imaginera que ça puisse être efficace. Pourtant, c'est ce qui se passe dans un logiciel, et sans aucun mécanisme d'autocorrection : l'ordinateur fait très bien ce qu'on lui dit, et quand une instruction est mauvaise, à la différence de ce qui se passe souvent dans l'entreprise, il l'exécute avec une conscience professionnelle absolue.

Autre caractéristique massivement sous-estimée : le coût de conception et de réalisation peut être élevé parce que, justement, l'objet est gros et invisible. En conséquence, il est toujours très difficile de savoir ce qu'il fait vraiment. De plus, il est très dur de savoir ce que l'on veut vraiment lui faire faire, et de l'exprimer dans les moindres détails. C'est donc compliqué de faire des logiciels qui marchent et la vraie bataille est celle de la prévention. Or, c'est délicat, surtout quand il faut expliquer aux gens, avant qu'ils ne se soient fait mordre, qu'il faut se méfier des petites économies. Certains espèrent qu'avec l'intelligence artificielle on pourra bientôt disposer d'un interlocuteur moins stupide. Mais la science ne suit pas et l'intelligence artificielle a dû très largement réduire ses prétentions.

Comment rater un logiciel

Il y a quantité de façons de rater. Une caractéristique fondamentale du logiciel est très largement sous-estimée : sa très grande instabilité intrinsèque. Fondamentalement, c'est un texte très long, écrit dans un langage spécifique, qui s'adresse à un interlocuteur obéissant mais qui ne comprend rien. La différence entre un logiciel qui marche et un qui ne marche pas, c'est epsilon : deux caractères en trop le transforment complètement. En mécanique, on peut jouer avec les tolérances, pas dans le logiciel. On ne peut qu'écrire l'ensemble des détails et si on en oublie quelques-uns, on en subit les conséquences. Il y a des raisons mathématiques profondes à cela ; mais en général on n'est pas habitué à comprendre ce phénomène, car il faudrait être familier de la logique, qu'on n'apprend pas à l'école, et qui répond à des lois très différentes des mathématiques classiques.

De plus, on n'a aucun moyen de simuler entièrement les comportements d'un logiciel, qui peut en avoir un nombre astronomique. Si on le lui demande, l'ordinateur fera chaque comportement en un temps très faible, mais on ne peut pas les essayer tous. De plus, l'intelligence humaine est assez désarmée par rapport à l'analyse de tous ces comportements. On peut donc rater une informatisation même en n'ayant que des gens très intelligents sur tout le parcours, si on ne dispose pas d'outils d'analyse adéquats.

Faire un logiciel, c'est un combat permanent contre trois ennemis bien armés : le "bug", la divergence et la fossilisation.

Le "bug"

L'ennemi numéro un, c'est le bug[2]. C'est une erreur mineure qui peut parfois provoquer des catastrophes majeures :
- THERAC 25 : le bug consiste en une série d'erreurs classiques dans l'interface homme/machine d'un appareil de traitement anti-cancéreux ; il a tué pas mal de gens et, bien que connu, n'a pu être corrigé pendant très longtemps ; des modifications successives ont été faites, mais fausses elles aussi ;

[2] En anglais, ce terme désigne un insecte. En français, le mot retenu est *la bogue* qui désigne l'enveloppe de la chataîgne.

- AT&T : un bug a provoqué le crash de l'ensemble du téléphone longue distance américain pendant une journée ; il était dû à un mauvais placement de parenthèse ;
- PATRIOT : la chute d'un missile Patriot sur une caserne américaine à Dahran est due à un bug ; c'est peu connu ;
- ARIANE : c'est un bug assez étonnant dont je parlerai plus loin en détail.

Le plus important, dans le bug, c'est son coût disproportionné. C'est donc l'ennemi n° 1. Mais personne au monde ne peut dire qu'il a chassé tous les bugs d'un logiciel.

La divergence

Le deuxième ennemi, assez fréquent, c'est la divergence : on n'arrive pas au bout du projet, tout en y ayant investi des sommes astronomiques. Elle a l'avantage de ne pas créer de bugs, mais elle est quand même très dangereuse parce qu'elle en induit de manière insidieuse : quand on n'arrive pas à terminer un système, on est souvent obligé, au dernier moment, de le remplacer par un autre, pas forcément beaucoup plus solide. Quelques exemples :
- FUTURE SYSTEM : G. Dréan décrit dans son livre un grand projet d'IBM, qui diverge pour des raisons essentiellement informatiques ;
- NEWTON : c'est un projet qui a divergé, parce qu'Apple est parti sur une idée fausse : croire qu'un ordinateur arriverait à reconnaître suffisamment bien l'écriture manuelle ;
- TAURUS : ce projet d'informatisation de la bourse anglaise a été arrêté après avoir coûté 3,5 milliards de francs ;
- P20 : un système de contrôle de centrale nucléaire, coûtant plusieurs centaines de MF aussi, et qui est parti en divergence complète ;
- l'aéroport de Denver, qui est dans l'incapacité de bien gérer les bagages.
Parfois ce sont de très gros systèmes qui divergent, parfois des petits. Hors de l'informatique, des systèmes divergent aussi ; mais quand c'est le cas dans les travaux publics les gens sont scandalisés alors qu'on ne se trouble guère dans l'informatique. Et on peut recommencer.

La fossilisation

Le troisième ennemi, beaucoup moins connu mais très insidieux, est la fossilisation. Quand on a réussi à venir à bout d'un logiciel, en général après avoir surmonté des bugs et la divergence, on n'y touche plus ! Pourtant, un système informatique vieillit. Un exemple est celui du contrôle aérien aux USA qui se fait avec des vieilles machines IBM spécialement développées pour cette application. Le logiciel n'a pas été conçu comme transportable, les pièces de rechange des machines manquent et il y a des pannes de longue durée à répétition. La situation est grave car le système de remplacement est en divergence ! Le problème était connu, beaucoup de gens avaient tiré les sonnettes d'alarme, mais ils ont quand même été pris de court.

Le passage à l'an 2000 en gestion est un problème typique de fossilisation. On ne s'est aperçu que relativement tard que c'était un tel problème : beaucoup de logiciels auront du mal à passer l'an 2000, leurs dates n'ayant été codées que pour les deux derniers chiffres. Et pour certains, on ne sait simplement plus comment ils tournent, comme si on avait perdu la notice !

Ces trois grands ennemis doivent être pris très au sérieux, malgré les résistances, qui sont en partie d'ordre culturel.

Les âges de l'informatique

Une civilisation se définit autant par ses outils que par sa culture : on parle de l'âge de la pierre taillée ou de l'âge du bronze. De même, l'informatique a traversé plusieurs époques, mais elle ne les a pas franchies en bloc : on vit encore par endroits à l'âge de la pierre.

La première époque est celle de la *ligne tapée* : on fait tout à la main. C'est la grande informatique des années 1970. On écrit du code directement et on ne gère pas grand-chose. Quand on doit faire des modifications, on y va de manière approximative et, tant que le client est content, on n'a pas trop de raisons de s'appesantir, d'où bugs, divergences et fossilisations. Les informaticiens expliquent

toujours que ce qu'ils font marche, mais leurs patrons se sont dit : *« Il faut cesser de croire ce que disent les informaticiens. Ils ne savent pas gérer leurs systèmes. Il faut passer à autre chose »*.

Est alors arrivée l'ère de la *ligne gérée*, qui s'est considérablement développée avant de décliner aujourd'hui. L'objet étant immatériel, il est difficile à gérer et on a utilisé les mêmes méthodes qu'en aéronautique : on a mis en place une administration puissante, pour gérer les modifications, les versions, les spécifications, etc. Beaucoup de choses ont été améliorées mais cela restait toujours inopérant contre les bugs : dans les outils de gestion il y a beaucoup de formulaires mais un formulaire, par définition, ne s'intéresse jamais au contenu. Dans le cas d'Ariane, par exemple, l'ensemble de la gestion est irréprochable.

Aujourd'hui, on va vers l'ère de la *ligne engendrée.* On essaye d'avoir de véritables machines-outils : certaines lignes de code ne sont plus écrites par quiconque mais par des outils. Ça ne résout pas les problèmes durs mais ça permet de les simplifier en réduisant la taille des objets manipulés par les hommes et en augmentant leur *densité intellectuelle.* Ce point est essentiel : le logiciel reste avant tout une création intellectuelle, et il faut toujours être intellectuellement armé pour l'aborder. Parmi les nouveaux outils, il y en a d'extrêmement performants qui permettent, par exemple, de faire des vérifications très intensives. La CAO de circuit est l'exemple d'un domaine qui a été très vite et très bien outillé.

L'art d'échouer de manière logique

Les gens qui décident sur l'informatique n'ont souvent pas intégré ses spécificités dans leur culture. Si la culture est ce qui reste quand on a tout oublié, que reste-t-il si on n'a pas appris ? Bien des acteurs sont désarmés et n'ont ni l'envie, ni le temps de s'armer. Et quand on est désarmé, soit on s'intéresse aux détails, soit on prend des positions philosophiques et, partant de là, on tire des conclusions logiques. Voici un petit florilège de ces positions philosophiques et de leurs conséquences logiques.

« Le client ne sait pas ce qu'il veut, pourtant on le lui fait ». On trouve de nombreux exemples de cette attitude, cause de multiples divergences. Ce genre de position ne devrait normalement plus durer.

« Le client sait ce qu'il veut, mais c'est impossible ». C'est le cas du Newton. On fait des choses sans rapport avec l'état de l'art parce que, quand on n'a pas la culture - et même quand on l'a - on fait mal la différence entre ce qu'on veut et ce qu'on peut. En informatique vouloir et pouvoir, ce n'est pas pareil.

« Le client tient à ce que son problème soit original ». On voit souvent cela dans l'informatique embarquée. Quand je dis : *« On sait déjà faire telle ou telle chose »*, on me répond : *« Ah ! Mais chez nous, ça n'est pas pareil, c'est de l'embarqué* ! *»*.

« On fait tout à la main parce qu'on n'a pas confiance dans les outils ». Pourtant, quand on travaille sur le sujet, on a vite beaucoup plus confiance dans les outils que dans sa propre main ! Un outil bien fait, ça marche bien, ça peut tester soigneusement et surtout, ça factorise[3].

« On embauche des gens mal formés parce qu'ils coûtent moins cher ». Cela conduit à l'idée : *« Ce n'est pas la peine de mettre des gens très compétents parce que les clients ne savent pas faire la différence »*. Cette conception est cependant en voie de disparition.

« On rajoute des gens car on est en retard ». Ça a tué beaucoup de systèmes. Un livre magnifique, *The Mythical Man-Month*[4], décrit le développement des systèmes chez IBM, dans lequel les auteurs étudient très finement la divergence de certains projets.

---

[3] C'est-à-dire qu'à chaque utilisation de l'outil, on bénéficie des efforts qui ont été faits pour le perfectionner. [NDLR]

[4] *The Mythical man-month : essays on software engineering*, F.-P. Brooks, Addison-Wesley, Reading, Massachusetts, 1975.

« *On laisse faire l'interface homme/machine par les informaticiens* ». Je trouve qu'il est scandaleux de donner au grand public des machines beaucoup trop compliquées parce que conçues par des informaticiens. Le plus grand progrès qu'on puisse attendre des informaticiens, c'est qu'ils s'intéressent à leurs utilisateurs plus qu'à eux-mêmes.

« *On ne touche pas un logiciel qui marche »*, prise de position et conséquences dont on va parler en détail.

« *Le logiciel est fiable à 96 % »*. On entend beaucoup cela dans l'industrie. Si on pose la question de la fiabilité du logiciel à un informaticien et qu'il répond : « *La question n'a pas de sens bien précis. On ne sait rien dire sur la notion de probabilité de bugs : c'est un phénomène erratique »,* ses interlocuteurs ne le croient pas. Et pourtant, c'est vrai en grande partie. En mécanique, on sait dire qu'une pièce va casser au bout d'un certain temps, mais en informatique, les phénomènes sont très discontinus.

Comment faire face et créer des systèmes qui marchent ? Je pense qu'il faut d'abord développer le *bon sens informatique*. Même sans trop connaître le sujet, on peut analyser les erreurs commises, baliser les chemins qui se sont avérés les plus sûrs, repérer les constantes et éviter de reproduire les mécanismes qui conduisent à des erreurs certaines.

Le bug d'Ariane

Le cas de l'accident d'Ariane V est intéressant et je peux en parler car un rapport d'enquête détaillé a été publié. Une fusée est une espèce de balai mou poussé du bas par un gros moteur. L'objet n'est pas facile à stabiliser mais des automaticiens très sérieux travaillent sur les équations de stabilisation et y parviennent. Leurs équations sont mises sur informatique et un ordinateur contrôle le tout. Dans Ariane V, les données nécessaires à ces calculs sont fournies par des gyrolasers, qui renvoient périodiquement un angle par rapport à la verticale, mesure fondamentale pour stabiliser la fusée. Il y a quelques milliers de lignes de code, pas spécialement difficiles dans ces gyrolasers, mélanges de physique et d'informatique. Il ne s'agit bien sûr pas de dire que ceux qui on fait Ariane sont mauvais, ce qui n'est pas le cas. Simplement, certaines façons de penser conduisent à des catastrophes, il faut le savoir. Cette panne était due au concours de quatre circonstances particulièrement révélatrices.

1°- Dans un logiciel d'Ariane IV, une séquence spéciale permet d'éviter un recalage des gyrolasers à chaque arrêt du compte à rebours, ce qui serait une opération très lente. Le programme écrit dans une mémoire permanente une série de valeurs, qui sont ensuite reprises au redémarrage de la chronologie. Ça n'a servi qu'une fois pour un lancement réel. Pour Ariane V, dont la procédure de lancement est différente, c'était sans la moindre utilité, mais on a repris le même logiciel. Pourquoi ? *Parce qu'on ne change pas un logiciel qui marche*. Une des variables sauvegardées est la vitesse horizontale, qui est bien sûr très faible au sol.

2°- Après le décollage, cette séquence tourne encore, bien que totalement inutile : la vitesse horizontale continue à être sauvegardée. Pourquoi a-t-on choisi cette option ? Parce qu'on a pensé que c'était plus simple et plus sûr de laisser tourner ce bout de code que de l'arrêter. Je ne sais pas d'où vient cette analyse, mais elle n'est pas forcément idiote.

3°- La vitesse horizontale, qui ne sert à rien ni à personne, est calculée sur 32 bits flottants, mais stockée en mémoire sur 16 bits fixes. On doit donc faire une conversion et, si la valeur enregistrée tient dans 16 bits, ça marche, sinon ça fabrique un *overflow* (dépassement de capacité du registre).

4°- Des gens se sont demandé ce qui se passerait en cas *d'overflow.* Ils ont ajouté des tests de protection mais pas sur l'ensemble des variables. Est-ce pour ne pas perdre en performance ? On n'aurait pas protégé un code pour ne pas perdre en performance alors qu'il suffisait de l'enlever pour gagner en performance !

Lors de son lancement, Ariane V a pris normalement sa trajectoire mais, plus puissante qu'Ariane IV, sa vitesse horizontale a augmenté plus rapidement. Au bout d'un moment, cette variable a été écrite en

32 bits dans du 16 bits et le processeur a reçu un signal de dépassement de capacité, ce qui n'était jamais arrivé à Ariane IV, plus lente et avec des trajectoires différentes.

Comment une anomalie banale dans un bout de code inutile a-t-elle pu faire perdre la fusée ? C'est là que la culture entre en jeu. Une loi dans le spatial déclare qu'une panne n'est pas grave si elle est détectée. C'est parfaitement rationnel en mécanique où ce qui est important, c'est de détecter si un système est en panne pour pouvoir basculer sur les systèmes de secours. Mais dans le cas présent, on a une projection sur l'informatique des lois d'un autre domaine. L'informatique du premier gyrolaser détecte *l'overflow ;* la main est alors donnée à un bout de code qui écrit en clair « *overflow ligne tant, programme machin »* et qui envoie à l'ordinateur principal un message déclarant en substance : « *J'ai détecté ma panne, je m'arrête, j'ai fait mon travail, je suis content »*. La deuxième centrale, identique, fait la même chose, envoie le même message quelques millisecondes après. Les deux gyroscopes sont alors déclarés en panne - alors qu'ils marchent parfaitement - et la fusée est perdue. Le bug étant déterministe, on aurait pu lancer cent fusées, elles auraient toutes éclaté au même endroit.

Ce qui est intéressant, c'est que l'existence de ce bug est liée à des phénomènes dont les gens avaient été prévenus, mais leur culture ne les prédisposait pas à entendre ces avertissements. Les recommandations de la commission d'enquête le montrent d'ailleurs. La première est de ne pas faire marcher un logiciel qui ne sert à rien. La deuxième est de faire des tests avant le lancement avec la vraie trajectoire. L'impasse avait été faite sur ces tests, sans doute à cause de leur coût. La commission recommande surtout de ne pas considérer qu'un logiciel marche sans défauts, ce qui veut donc dire que *les bugs de logiciels n'étaient pas considérés comme critiques jusque-là !*

De fait, il y a très peu d'informaticiens au CNES. On y trouve des mécaniciens et des aérodynamiciens très compétents mais qui, comme tout un chacun, projettent leur culture alors que l'informatique n'a rien à voir avec ce qu'ils maîtrisent. Ainsi, lorsqu'intervient une panne dans le matériel, la plupart du temps, c'est sur quelque chose qui marchait bien avant : il s'use. Le logiciel, lui, ne s'use pas : s'il apparaît une erreur, c'est qu'elle y était depuis le début. C'est peut-être pour ça qu'on est tenté de croire que le logiciel est fiable à 100 %.

Beaucoup de gens demandent alors : « *Pourquoi ne fait-on pas de systèmes redondants en informatique ? »* L'Airbus A320 est bien fait comme cela ! En fait, c'est très délicat et deux questions se posent. Si on met deux codes au lieu d'un, on va payer deux fois plus cher le développement ; est-ce la meilleure façon de dépenser l'argent ? Et si à un moment les deux codes ne disent pas pareil, que fait-on ? J'ai vu des cas où on prenait celui de gauche : eh oui, il faut bien dire quelque chose !

J'ai aussi vu des spécifications d'organes de sécurité de centrales nucléaires prescrivant ceci : « *Dans le cas d'arrivée d'alarmes nombreuses et rapprochées, le logiciel devra réagir en conséquence »*. Cela veut dire qu'on ne sait pas analyser le problème, et qu'on abandonne devant la difficulté (heureusement, c'est un exemple ancien).

La dernière recommandation de la commission d'enquête Ariane, très importante pour le monde du logiciel mais peu connue, est : « *Prière de simplifier d'un ordre de grandeur l'organisation de production de logiciels qui fait appel à des sous-traitances multiples ».* Faire faire des logiciels critiques par des gens répartis chez de multiples sous-traitants qui ne s'entendent pas forcément bien n'est pas sans risques.

Alta Vista

Par contraste, je prendrai comme exemple de ce qui marche Alta Vista. Le problème dans Internet est d'y trouver ce qu'on cherche. Alta Vista est un outil qui donne la liste des documents dans le Web répondant aux critères que vous lui avez donnés, avec un temps de réponse hallucinant d'une seconde environ. C'est un outil indispensable pour tous ceux qui travaillent avec Internet. On a déjà indexé plus de soixante millions de documents dont certains peuvent être longs. Alta Vista, comme ses concurrents, cherche donc parmi plusieurs milliards de mots et le nombre journalier de questions auxquelles il répond est de l'ordre de vingt millions.

Pour faire cet objet-là, on a dû briser beaucoup de tabous. En premier lieu, il a fallu mettre une algorithmique très sophistiquée, jalousement gardée secrète. Ensuite, un tabou très prégnant stipulait qu'une base de données devait être sur disque. Dans Alta Vista, pour atteindre les performances requises, on mobilise plusieurs machines de dix processeurs et douze gigaoctets chacune, et on rajoute des machines pour suivre la croissance d'Internet. Il a fallu, pour concevoir cela, surmonter l'incapacité à anticiper le doublement des performances : 100 gigaoctets de mémoire sur une machine, ce sera chose possible dans deux ans, il suffit d'attendre !

Alta Vista a également suscité une logique de commercialisation nouvelle. On a commencé par y donner un accès gratuit, tout comme Georges Eastman avait donné deux cent cinquante mille appareils photo aux enfants des écoles. Alta Vista a été conçu par deux chercheurs de chez DEC, qui travaillaient sur des processeurs nouvelle génération. Leur projet ayant été arrêté, ils se sont consacrés à Alta Vista dans leurs labos. Pourquoi le donnent-ils alors ? Parce qu'ils s'attaquent en fait à un marché qui n'est pas du tout celui d'Internet mais celui de l'Intranet[5]. C'est-à-dire qu'en permettant à toutes les entreprises de consulter leur fonds de données amorphes, des armées de textes et de procédures, et d'en tirer quelque chose, ils proposent des services inestimables et inconnus. Netscape, puis Microsoft, ont de même distribué leurs navigateurs Web quasi gratuitement, en accroche de leur produit principal : les serveurs.

DÉBAT

Un intervenant : *Alta Vista est une simple liste, peut-être d'une longueur inhumaine, mais avec un début et une fin. Alors qu'Ariane c'est très compliqué.*

Gérard Berry : Contrairement à cette idée naturelle, le logiciel d'Ariane est beaucoup plus simple qu'Alta Vista. Ce sont des équations mathématiques qu'il s'agit de calculer.

Int. : *Les gens du CNES ont des procédures et des méthodologies rigoureuses. À l'inverse Alta Vista évoque des bidouilleurs. Suggérez-vous que l'excès de méthodologie soit parfois cause d'erreurs ?*

G. B. : Le CNES avait demandé une vérification de ce bout de code, pour s'assurer qu'il ne pouvait pas y avoir d'erreur à l'exécution du type *overflow.* Elle a bien été effectuée, sauf sur la ligne en question. C'est pourquoi la commission, dit : « *Quand on commande une vérification, il ne faut pas seulement s'intéresser au fait qu'elle a été faite, mais aussi à ce qui a été vérifié* ». C'est la limite des méthodes : quand on n'en a pas, on va droit à la faute. Mais quand on en a, ça ne dispense pas de s'intéresser au contenu. Un des grands problèmes de ce domaine, c'est qu'il y a des méthodes et des outils - par exemple, dans Alta Vista il y a des outils, dans Ariane, des méthodes - mais pour l'instant, ce sont deux sujets disjoints.

Int. : *Il faut bien faire confiance à celui qui revient en disant :* « J'ai tout contrôlé ; c'est bon », *sinon le problème sera vite* : « Qui contrôle le contrôleur du contrôleur ? ».

G. B. : Beaucoup d'échecs sont dus au fait que des gens de bonne foi ont dit que ça marchait. Les mathématiques sont connues comme un domaine rigoureux. Mais, même quand un mathématicien écrit un papier, on ne lui fait pas confiance a priori. On le fait relire par d'autres mathématiciens, et les éditeurs relisent les rapports d'évaluation. Celui qui commande une vérification se doit de relire ce qu'on lui donne. Sur des logiciels qui mettent en jeu des sommes aussi importantes, c'est indispensable.

La chasse aux bugs

Int. : *Naguère, à la RATP une commission de sécurité était chargée de contrôler la fiabilité des nouveaux systèmes. Avec la technologie des relais électromagnétiques, elle pouvait identifier tous les*

[5] Réseaux internes à l'entreprise, offrant les mêmes fonctionnalités qu'Internet, mais inaccessibles par les personnes qui ne sont pas habilitées par l'entreprise. L'utilisateur découvre la puissance de l'outil de recherche en l'utilisant gratuitement sur Internet, et voit l'intérêt de disposer du même confort sur son réseau Intranet.

cas de figure. Avec le RER B automatisé, on est passé aux logiciels. Et là, elle a bloqué, retardant la mise en service de la ligne. Le ministre des Transports s'en est inquiété et on a été chercher Abrial, un spécialiste de la question, qui a dit : « Voilà comment on fait ailleurs ». *Mais la commission n'a pas compris, c'était trop compliqué pour elle. Finalement, le ministre a tranché en faveur des préconisations d'Abrial et la commission a déposé les armes.*

G. B. : La technique d'Abrial était utile à une époque où on ne pouvait analyser les centaines de millions de configurations possibles d'un système. Elle reste utile, mais d'autres technologies d'analyse exhaustive sont apparues. Intel, par exemple, avec le bug de son Pentium qui lui a coûté 300 millions de dollars d'après ce qu'on dit, cherche à engager les meilleurs spécialistes de la vérification de circuits et de programmes. Ce qu'on n'imaginait pas pouvoir faire il y a deux ans est aujourd'hui fait en quelques secondes ou minutes, à cause de progrès scientifiques majeurs.

Int. : *On sait chasser les bugs alors ?*

G. B. : Dans des domaines assez précis et à condition d'employer des technologies de fer. Définir ce que doit faire Ariane, ce n'est pas très compliqué. Là où on sait le mieux chasser les bugs, c'est dans les protocoles de communication. La spécification y est relativement facile et on arrive à faire des vérifications. Définir ce que fait un gros logiciel de gestion, c'est une toute autre affaire…

Int. : *On l'a pourtant trouvé sur Ariane V ?*

G. B. : Ce bug a été trouvé et réparé, mais après l'explosion ! Avant tout, il faut se donner les moyens d'éviter l'échec, en évitant par exemple la sous-traitance massive. Les avionneurs utilisent une informatique sophistiquée et rigoureuse, mais pour les gens de l'Aérospatiale, dans une fusée, il n'y a pas vraiment d'informatique, voilà un problème culturel : même s'il n'y a que très peu de code, ça peut être dangereux. Mais ça change car on ne peut plus se permettre de faire ce genre d'erreur.

Informatique de gestion et bugs

Int. : *Un vieux routier de l'informatique définissait deux approches en matière de logiciel : la Massachusetts Approach, celle du MIT, et la New Jersey Approach, celle des Bell Labs. La première, c'était le système Multics, l'un des meilleurs logiciels du monde dans les années 60 avec dix bugs répertoriés, équipant en particulier les machines de la NASA. Il a aujourd'hui disparu. Dans le même temps, les Bell Labs développaient le système Unix, griffonné par deux ingénieurs avec de nombreux bugs. Aujourd'hui Unix tourne sur énormément de machines. On peut faire de même un parallèle entre le Mac, typiquement Massachusetts Approach et Windows, typiquement New Jersey approach : la première version de Windows était très mauvaise, pleine de bugs et largement insuffisante face aux standards du Mac, et pourtant, là aussi, c'est Microsoft qui a gagné et Macintosh qui est réduit à une présence de niche. Face à ce genre d'exemple, en informatique de gestion, ne peut-on dire qu'il ne faut pas trop se soucier des bugs en laissant aux utilisateurs le soin de les découvrir ?*

G. B. : Non, dès sa naissance, Unix était beaucoup plus simple que Multics et avait peu de bugs. Unix est d'ailleurs très daté, difficile à faire évoluer et pas aussi intelligent que ça, avec un interface homme/machine médiocre. Quant à Windows, Bill Gates n'a laissé à personne d'autre le soin de faire la première démonstration de Word : avec, à l'époque, 600 bugs reconnus dont 100 plantaient la machine, il a réussi sa démonstration avec talent en les évitant tous. Mais Bill Gates est un génie de la démonstration.

Int. : *Dans les grands systèmes de gestion, on ne peut empêcher les utilisateurs de changer d'avis en cours de développement, et ils ne sont pratiquement pas spécifiables.*

G. B. : Effectivement, et de plus on ne connaît d'avance ni la variété de configurations du matériel, ni leur environnement en logiciels d'application. La plupart du temps, le bug ne se révèle que dans tel cas précis, avec telle carte graphique implantée à mille exemplaires dans le monde, par exemple.

Autre grosse différence : comment tester ? Dans un logiciel de type Ariane, on ne peut tester en environnement réel, d'où l'importance des outils, heureusement très fiables. Dans un domaine de type

Alta Vista, on peut tester en environnement réel parce que, si le système ne donne pas les bonnes réponses au premier utilisateur, ce n'est pas catastrophique. Donc, la meilleure façon de tester est souvent de mettre le logiciel entre les mains d'un grand nombre d'utilisateurs qui détecteront les erreurs. Ce sont les bêta-tests, largement pratiqués pour les grands logiciels du commerce.

Int. : *Le partage de la réalisation d'un grand système entre de multiples sous-traitants n'est pas l'apanage de l'informatique. Où est la spécificité du logiciel ?*

G. B. : Quand, dans un grand programme, on commence par établir les liens d'argent entre les divers partenaires avant de savoir ce qu'on va faire, cette logique financière est dangereuse. Mais il n'est pas impossible qu'elle change, surtout en payant aussi cher les erreurs : le vrai problème d'Ariane n'est pas que la première ait explosé, c'est qu'il n'est pas possible que la seconde explose.

Bien sûr, il y a des bugs ailleurs, en mécanique ou dans les procédures administratives. L'accident de la Gare de Lyon est en partie dû à l'équivalent d'un bug : la procédure dit qu'en cas de problèmes de freins, il y a les cas de figure A et B. Mais ce n'est qu'à la fin du cas B qu'elle précise que, dans les deux cas, le circuit de freins doit être purgé. Et le conducteur était dans le cas de figure A. C'est un bug de même nature que celui d'Ariane. Mais si dans un système mécanique bien construit le bug qui apparaît au niveau du système est une anomalie, c'est le cas général en informatique : c'est une différence majeure.

Int. : *Le client, c'est très souvent trois personnes : celui qui spécifie, celui qui utilise et celui qui paye. Elles ont souvent des conflits d'intérêts. Comment s'étonner de ces difficultés ?*

G. B. : Je ne connais personne capable de faire l'estimation du coût de développement d'un logiciel et les dirigeants d'entreprise ne peuvent souvent plus supporter cette explosion des coûts. Mais les choses changent. Pourquoi parler de l'embarqué ? Des plantages immenses, des projets divergents à très grande échelle, il y en a eu quantité de passés sous silence. Mais quand ça se passe en direct à la télévision, on en parle.

Vers une informatique sûre ?

Int. : *Vous avez parlé d'instabilité intrinsèque et j'ai cru comprendre que, pour des raisons mathématiques, la vérification du logiciel était sans espoir.*

G. B. : On est sans espoir pour la vérification des logiciels complexes par les hommes, mais pas pour la vérification par les mathématiques. Qu'est-ce que l'instabilité ? C'est le fait qu'une modification d'un morceau de code se propage et que l'effet est globalement incompréhensible. Mais dans certains domaines, on peut analyser exhaustivement les conséquences. Nous utilisons ces techniques dans la fabrication d'un langage (Esterel) pour des applications embarquées. Chaque fois qu'on fait une modification, on vérifie avec un vérifieur formel que certains comportements critiques de l'objet n'ont pas changé.

Int. : *Quelle est la nature de ce vérifieur formel ?*

G. B. : C'est un système de calcul booléen qui prend des formules logiques gigantesques et détermine si elles sont vraies ou fausses, ce qui est hors de portée pour un homme.

Int. : *Peut on imaginer une informatique "fail safe" (qui laisse le système dans une configuration sûre en cas de panne) ?*

G. B. : On sait encore assez peu de choses sur les concepts liés aux logiciels. On sait qu'on n'aura pas de langage de programmation vraiment bien fait d'ici quinze ans, parce qu'il faut au moins quinze ans pour en développer un et qu'il n'y a pas de candidat déclaré.

Un des gros problèmes en informatique, c'est qu'il y a souvent plusieurs architectes pour un programme, alors qu'on pourrait travailler mieux avec moins de gens mieux formés. Un circuit comme le Pentium est beaucoup plus complexe qu'un logiciel. Pourtant, il est fait par très peu de personnes, quatre ou cinq pour le coeur et des centaines pour les périphériques, et les tests atteignent 75 % du

coût. Ce sont des processus convergents. Dans le logiciel on n'a pas encore compris l'utilité de ce genre de processus.

On ne sait pas en réalité comment attaquer le problème de l'informatique "fail safe", même s'il y a beaucoup de littérature sur le sujet. On peut s'en approcher en faisant de la redondance et beaucoup de tests. Mais le "fail safe" ne marche que dans un milieu où il y a de la continuité, ce qui n'est pas le cas ici.

Des génies bondissants

Int. : *À quoi ressemblent les génies de l'informatique ?*

G. B. : Les Américains disent qu'ils aiment bien travailler avec les Européens, que se sont des gens créatifs, qui connaissent des mathématiques et plein d'autres choses qu'eux ne connaissent pas, mais… que ce ne sont pas des concurrents. Tout se passe dans la Silicon Valley et autour du MIT. La Silicon Valley est un endroit étonnant, avec des taux de croissance qu'on n'imagine pas. Synopsys, le leader de la synthèse de circuit, fait des choses très high tech avec des gens de toute première force : ils étaient quatre en 1987, aujourd'hui ils sont deux mille.

Int. : *Sur quoi travaillent-ils aujourd'hui ?*

G. B. : On voit des changements de paradigme très brutaux. Avant c'était : « *iI n'y a pas assez de transistors dans les circuits, donc il faut qu'on les serre au maximum »*. Depuis un an, changement complet de paradigme : « *Que faire de tous ces transistors dans nos circuits ? ».*

Pour les firmes le *time to market*, est essentiel car elles vivent sur des marchés en croissance exponentielle : au-dessus de l'exponentielle, on pense qu'on est à l'abri, mais ça ne dure pas. La vie des firmes, c'est cela : pour survivre, il leur faut toujours être au-dessus de l'exponentielle. Les gens sont prêts à prendre des paris. Pour eux, avoir des gens de grande classe avec un bon PhD est prioritaire. Et ils ont beaucoup d'argent.