https://laurentbloch.org/BlogLB/Comment-travailler-avec-des

Comment travailler avec des informaticiens ?

- Zinformatiques - L'informatique : science et industrie - Systèmes d'information -

Date de mise en ligne : mardi 4 avril 2006

Sommaire

- Chapitre 6 Comment travailler avec des informaticiens ?
  - Échapper au développement
  - Un logiciel existe déjà
  - Travailler sur deux plans orthogonaux
  - Des logiciels moyens pour des systèmes sobres
  - Le modèle de l'équipe de tournage
  - Le logiciel libre, phénomène technique, social et culturel
  - Le modèle Java, et autres types de composants
  - Et s'il faut malgré tout développer ?
  - Conditions initiales indispensables
  - Équilibre entre maîtrise d'ouvrage et maîtrise d'oeuvre
  - Relations de travail informatiques
  - À quoi les informaticiens passent-ils leur temps ?
  - Laconiques leçons de l'expérience

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">
Comment travailler avec des informaticiens ?

Ce texte est extrait de mon livre Systèmes d'information, obstacles et succès, paru en mars 2005 aux Éditions Vuibert.

Page d'accueil de ce livre Chapitre 6 Comment travailler avec des informaticiens ? Si nous essayons de résumer ce que nous avons écrit au fil des quatre premiers chapitres de ce livre, le bilan ne semble pas très encourageant pour un maître d'ouvrage qui voudrait lancer un projet d'informatisation en faisant appel à des informaticiens, que ce soit ceux d'une équipe interne ou d'une société extérieure :

1. définir le travail à faire, c'est-à-dire rédiger un cahier des charges et des spécifications détaillées, est très difficile, sinon impossible ;
2. la plupart des méthodes mises au point à cet usage sont lourdes, onéreuses et inefficaces, sinon contre-productives ;
3. les informaticiens ont une tournure d'esprit et une psychologie assez particulières qui rendent le travail avec eux très difficile et incertain ;
4. la complexité du travail des informaticiens en rend l'encadrement et le contrôle également difficiles et incertains ;
5. les seules méthodes qui semblent prometteuses demandent un investissement personnel très important de la maîtrise d'ouvrage, et ce tout au long du projet ;
6. ces méthodes prometteuses sont pour la plupart de surcroît itératives, car l'objectif se définit progressivement au cours de la réalisation, avec la conséquence souvent perçue comme déplaisante que l'on ne sait en commençant ni combien de temps cela va durer ni combien cela va coûter (la réponse à cette dernière objection, c'est qu'avec une méthode classique et un cahier des charges, on ne sait pas non plus combien de temps cela va durer ni combien cela va coûter, mais que l'on en a l'illusion).

Bref, il faut être inconscient pour se lancer dans une telle aventure. Que faire alors si on a vraiment besoin d'un système informatique ?

Échapper au développement

Un logiciel existe déjà

La première question à se poser est : n'existe-t-il pas un système déjà réalisé qui pourrait convenir ? Si la réponse est oui, et si le logiciel est disponible, il faut l'adopter sans hésiter : quel qu'en soit le prix ce sera moins cher, moins long et moins douloureux que de se lancer dans un développement *de novo*.

Bien sûr, pour déterminer la réponse à la question ci-dessus il ne faut pas oublier la proposition « qui pourrait convenir ». Cela doit être examiné avec soin. Le logiciel correspondant peut être un progiciel du commerce ou un logiciel libre. S'il ne convient qu'à 90%, il est presque certainement préférable de s'adapter soi-même au logiciel que l'inverse : il faut en effet renoncer à la tentation d'adapter le logiciel à ses besoins, parce qu'une adaptation c'est un développement, et donc plonger à nouveau dans la situation horrible décrite au début de ce chapitre. Un ego sur-dimensionné, ou le respect inopportun de procédures issues du passé, ou l'obéissance aux caprices de certains responsables conduisent certains maîtres d'ouvrage à demander de multiples adaptations aux progiciels qu'ils achètent ; ce sont autant de développements spécifiques qui vont grever le budget jusqu'à atteindre le coût d'un développement entièrement nouveau, tout en détruisant la cohérence initiale du logiciel ; il convient donc de les limiter au strict nécessaire, et plutôt profiter de l'adoption du nouveau système informatique pour réformer les procédures de gestion.

Une variante de cette solution pourrait être qu'il n'y ait pas un logiciel existant qui réalise la fonction voulue, mais plusieurs logiciels qui en réalisent des parties. L'assemblage de plusieurs logiciels pour en faire un plus gros, c'est un développement, et ce développement ne sera un projet réaliste que si les caractéristiques techniques des logiciels envisagés s'y prêtent. Les logiciels libres, de par l'ouverture de leur code source et leurs méthodes particulières de développement exposées à la section 6, se prêtent en général particulièrement bien à cet exercice, d'autant mieux qu'il existe pour ce faire des outils libres très bien adaptés comme les langages de script *Perl* et *Python*. Le développement envisagé ici sera sans doute peu volumineux, ce qui le rendra plus raisonnable.

Travailler sur deux plans orthogonaux

Des épisodes précédents de ce récit se dégage l'idée que ce qui rend si difficile et si hasardeux le développement d'un logiciel, ce sont les relations de l'équipe de développement avec le maître d'ouvrage, que nous pouvons assimiler de façon générique à un *client* : dès lors qu'un client veut donner des consignes à des développeurs, c'est compliqué et potentiellement conflictuel. Nous avons expliqué pourquoi la rédaction de cahiers des charges ne résolvait rien, et nous pouvons même dire que plus le cahier des charges sera précis et détaillé, plus le développement sera long, onéreux, conflictuel et au bout du compte décevant, tout simplement parce que le cahier des charges sera de toute façon non pertinent, et qu'une erreur détaillée et circonstanciée est plus difficile à corriger qu'une erreur brève et imprécise.

Nous avons évoqué ci-dessus une façon de résoudre ce problème : l'*eXtreme Programming* organise le

développement autour d'une équipe qui associe maîtrise d'ouvrage et maîtrise d'oeuvre de façon pratiquement fusionnelle. Le cahier des charges s'élabore de façon incrémentale au cours même de la réalisation, et au vu de ses résultats intermédiaires. Cette solution paraît très séduisante, et nous décrirons ci-dessous (page ??) une expérience, assez différente mais fondée sur un principe voisin, qui fut couronnée de succès. Lorsque c'est possible nous pensons qu'il faut au moins s'inspirer de cette méthode, sinon en appliquer tous les détails. Mais nous avons aussi souligné que la sociologie particulière des milieux de direction des entreprises et les rapports de pouvoir qui s'y nouent rendaient sans doute une telle démarche souvent impraticable.

Si la fusion entre l'équipe de maîtrise d'ouvrage et celle de maîtrise d'oeuvre n'est pas possible, on peut essayer une méthode qui en soit exactement le contraire : leur séparation totale. Observons d'ailleurs que c'est ce qui se passe si l'on a recours à un progiciel ou à toute autre forme de logiciel « tout fait », au paramétrage près, auquel de toutes les façons on n'échappera pas. Pendant les vingt ans durant lesquels j'ai dirigé des équipes d'informatique scientifique, c'est ainsi que j'ai procédé. Nous mettions en place des infrastructures (ordinateurs, systèmes, réseaux) que nous nous efforcions de rendre le plus accessibles et le plus disponibles possible, sur ces infrastructures nous installions les logiciels et les bases de données qui nous semblaient les plus appropriés aux usages des chercheurs, qui d'ailleurs ne se privaient pas de nous faire des suggestions. Nous organisions des formations à l'usage de ces systèmes, et de la sorte nous répondions à la plupart des demandes exprimées ou potentielles. De ces interactions avec les chercheurs naissaient des idées de logiciels, que nous développions de notre propre initiative, non sans bien sûr soumettre nos prototypes à la critique éclairée des utilisateurs. Les résultats n'ont pas été mauvais, comme en témoignent quelques documents [1].

Est-il possible d'obtenir par ce procédé un système de contrôle du trafic aérien ? un système de gestion financière et comptable ? Il n'y a pas de réponse générale à cette question, et cela dépend pour une grande part de l'équipe de développeurs, qui peut avoir en son sein des compétences financières et comptables, ou en contrôle de trafic aérien : j'ai connu un responsable informatique qui lisait quotidiennement le Journal Officiel pour adapter son logiciel à la réglementation, qu'il connaissait bien mieux que le directeur financier et le chef comptable. Signalons que beaucoup de logiciels de pointe sont constitués au moins en partie de travaux effectués à l'occasion d'une thèse universitaire ou d'un diplôme d'ingénieur, et ce dans les domaines les plus variés, y compris le contrôle du trafic aérien. On peut imaginer aussi que les chances de succès seront meilleures si les développements sont constitués, plutôt que d'un seul logiciel monolithique, de plusieurs logiciels de taille moyenne et fonctionnellement relativement autonomes : trésorerie, comptabilité, budget, commandes, gestion des antennes régionales, etc. Cela est possible sans pour autant renoncer à la cohérence des données. J'ai rencontré un dirigeant qui se plaignait de ne connaître la situation comptable de son entreprise (un établissement public) qu'avec un retard d'un mois : la cause de ce dysfonctionnement n'était sûrement pas technique, même si la technique y contribuait. Ce qui a été dit plus haut sur les règles de la comptabilité publique et sur l'organisation sociale des gens chargés de les appliquer me semble une meilleure piste pour en trouver les raisons.

Des logiciels moyens pour des systèmes sobres

De façon générale, tout plaide, lorsque c'est possible, pour le logiciel de taille moyenne. Le plus souvent un logiciel devient gros non pas par l'effet du volume et de la complexité des tâches qu'il doit accomplir, mais à cause de la lourdeur de l'organisation sociale du maître d'ouvrage, de la difficulté à résoudre les conflits entre les différents groupes concernés, et de la difficulté à comprendre le cahier des charges. L'intervention de prestataires extérieurs n'est finalement sollicitée que pour résoudre des conflits internes autrement insolubles, et elle contribue rarement à rendre le projet plus sobre. Si l'on peut s'affranchir de cette organisation en supprimant toute la démarche de cycle en V, de cahier des charges et de groupe projet, tout deviendra plus simple et le

logiciel pourra maigrir pour devenir moyen.

J'entends par moyen un logiciel dont la version initiale pourra être développée en moins de deux ans par une équipe de moins de huit personnes, maître d'ouvrage délégué et responsable de l'équipe compris. On note que cette taille d'équipe est compatible avec la méthode *eXtreme Programming*.

En fait, pour développer un tel logiciel, le mieux sera sans doute de commencer par un prototype développé en six mois par deux ou trois personnes. Ce prototype fera sans doute apparaître une structuration du logiciel telle que des grandes parties pourront être considérées comme relativement indépendantes, ce qui permettra une division du travail plus facile et donc un accroissement de la taille de l'équipe.

Supposons que nous soyons dans un cas où un développement de logiciel n'a pas pu être évité. Si le logiciel est moyen, c'est-à-dire sobre, et que le périmètre qu'il englobe est limité, le risque en laissant un peu la bride sur le cou à l'équipe de développement redevient raisonnable. Si cette équipe, au lieu d'être exclusivement constituée d'employés d'un prestataire extérieur, comporte des développeurs de l'entreprise maître d'ouvrage, le client pourra assurer la maintenance de l'application et organiser des itérations du cycle de développement pour converger vers ce que souhaite l'utilisateur, sans encourir pour autant des coûts faramineux de « tierce maintenance applicative ». Mais une telle démarche n'est généralement pas populaire parce qu'elle n'engendre pas les projets pharaoniques et les budgets colossaux qui en propulsent les responsables à la une de la presse professionnelle ; elle ne permet pas non plus de se débarrasser *complètement* de ses informaticiens. Il serait intéressant d'étudier comment les entreprises de type SSII ont inventé les « concepts » de la conduite de projet, de la tierce maintenance applicative, de la réingénierie de processus, de l'externalisation et quelques autres, comment la presse spécialisée a propagé la bonne nouvelle, et comment les maîtres d'ouvrage incompétents gobent toutes les modes qui passent, pour le plus grand bénéfice des SSII et des consultants divers et variés.

Bref, il n'y a pas de recette miraculeuse, mais des moyens pragmatiques d'échapper à l'échec promis à coup sûr par le cycle de développement en V.

Le modèle de l'équipe de tournage

Le développement de jeux vidéo est aujourd'hui le secteur moteur de l'industrie informatique, tant pour le matériel que pour le logiciel. Il a suscité les avancées les plus récentes dans des domaines aussi variés que les micro-processeurs pour micro-ordinateur, les processeurs spécialisés pour les cartes graphiques, le traitement du signal, le traitement du son, et les algorithmes les plus efficaces pour tirer parti de tous ces dispositifs.

S'il y a encore dans ce domaine quelques réalisations artisanales, le développement de jeux engage le plus souvent des investissements très importants. Le cycle de développement en V avec cahier des charges n'y est pas de mise, le processus ressemble plutôt à celui de la réalisation d'un film : il y a un scénariste, le réalisateur part du scénario, mais il arrive qu'il modifie celui-ci au cours du travail ; il est assisté de graphistes, d'ingénieurs du son, d'assistants chargés de la continuité, du montage, des effets spéciaux, etc.

Ce type d'organisation me semble mieux correspondre à la nature intellectuelle d'un développement informatique que le modèle taylorien et le cycle en V.

Le logiciel libre, phénomène technique, social et culturel

Le recours au logiciel libre, dont nous avons déjà abondamment parlé, pourrait n'être qu'un cas particulier de la section ci-dessus, « un logiciel existe déjà », mais en fait la nature particulière des logiciels conçus selon la méthodologie du libre induit en général des usages spécifiques. Le monde du logiciel libre, comme le monde Unix dont il est pour l'essentiel issu, se caractérise par sa sociologie et par ses traits culturels presque autant que par une technologie originale. Pour réussir une informatisation fondée sur des logiciels libres il convient de se familiariser avec tous ces aspects. Lire les revues, fréquenter les associations et les conférences qui s'y consacrent peuvent constituer une bonne initiation, ainsi que le moyen de rencontrer des spécialistes dont certains pourront peut-être rejoindre votre projet, ou en tout cas vous mettre sur la piste de solutions qui vous conviennent. Hormis ces aspects socio-culturels dont il faut avoir conscience pour ne pas échouer, le logiciel libre est un moyen parmi d'autres pour éviter d'entreprendre le développement d'un logiciel.

Le modèle Java, et autres types de composants

La caractéristique la plus novatrice et la plus intéressante du langage Java réside dans son immersion au sein de l'Internet : un client WWW (comme un navigateur, *Internet Explorer* ou *Mozilla*) peut interroger un serveur qui va lui envoyer en réponse un programme Java appelé *applet* 1 qui s'exécutera sur la machine du client, dans le contexte du navigateur. À l'inverse, un programme Java peut utiliser un objet ou une classe qui se trouve sur un serveur à l'autre bout de la planète. C'est possible parce qu'un programme source écrit en Java est traduit (compilé) vers un langage intermédiaire indépendant du processeur sur lequel il sera exécuté, et il sera exécuté par une machine virtuelle Java *(JVM)*. Il suffit donc de disposer d'une *JVM* (il y en a une dans chaque navigateur WWW, sur la plupart des postes de travail usuels, dans les téléphones portables et les assistants personnels récents, etc.) pour utiliser à loisir des programmes Java obtenus par le réseau. Cela s'appelle du code mobile, et c'est révolutionnaire.

Cette technique révolutionnaire a suscité des pratiques nouvelles. On y a vu un moyen d'atteindre l'objectif tant désiré et si peu atteint du génie logiciel : la réutilisation de composants logiciels. Et effectivement sont apparues des bibliothèques publiques de classes Java, comme les EJB[] *(Enterprise JavaBeans)* créées et diffusées par l'inventeur du langage, *Sun Microsystems*. Le propos de telles bibliothèques est de fournir des fonctions élémentaires prêtes à l'emploi pour servir à la construction de programmes plus vastes dans un domaine particulier, que ce soit la cryptographie ou la gestion de commandes. L'auteur du programme final n'a plus qu'à assembler selon les règles fixées des classes testées et documentées.

Le succès de cette entreprise a été non négligeable, mais inférieur cependant aux espérances. Les raisons de cette déception relative nous semblent être les suivantes :

- Le langage Java a soulevé un grand enthousiasme lors de son lancement, mais il est vite apparu qu'il était plus complexe et plus difficile à utiliser que ce que l'on avait cru. Il a subi la concurrence de langages moins puissants mais plus faciles à utiliser, comme Perl et PHP pour la programmation « côté serveur » et Javascript (qui malgré son nom n'a rien à voir avec Java) pour la programmation « côté client », c'est-à-dire dans le navigateur. Pour ces langages se sont créées et développées des bibliothèques de composants (CPAN pour Perl, Pear pour PHP). Perl et PHP peuvent facilement être incorporés au logiciel serveur WWW le plus répandu, *Apache*, ce qui les rend très pratiques à mettre en oeuvre pour la programmation WWW. Perl, PHP et Apache sont de surcroît des logiciels libres, ce qui a favorisé leur dissémination.
- L'inventeur du langage a voulu en garder le contrôle, notamment en modifiant les spécifications d'une

version à la suivante, en poursuivant devant les tribunaux les producteurs de versions hétérodoxes et en pratiquant (au moins dans les premières années) une politique de disponibilité de machines virtuelles qui négligeait certaines architectures matérielles. Ces pratiques ont détourné de Java certains développeurs.

- Il est apparu que développer un logiciel en utilisant à tout crin des classes obtenues par le réseau, mais d'origines parfois incertaines, pouvait conduire à une perte de maîtrise du développement, le programmeur ne sachant plus très bien qui fait quoi dans son programme. Assez vite il n'est plus possible de garantir la qualité du code produit, non plus que son homogénéité. Ces inconvénients existent aussi bien entendu pour les bibliothèques de composants destinés à Perl, PHP ou Javascript. Il s'agit donc d'une leçon de prudence retirée des premières expériences de réutilisation à grande échelle de composants logiciels.

Malgré les réserves énoncées ci-dessus, le recours à des bibliothèques publiques de composants logiciels est une technique à suivre.

La construction de logiciels selon les principes de la programmation par objets, en Java ou en C++, a fait l'objet d'une élaboration méthodologique intéressante qui n'est pas sans parenté avec le recours aux composants logiciels (et qui ne l'exclut pas) : les modèles de conception réutilisables (*« Design Patterns »*, ou *patrons de codage*)[] ; au lieu d'importer un composant, on programme en suivant un patron, au sens que ce mot a pour les couturières, et qui est la traduction exacte de *pattern*.

Et s'il faut malgré tout développer ?

Conditions initiales indispensables

Si aucun des recours énumérés ci-dessus n'a permis d'échapper à la perspective effrayante de devoir développer un logiciel *de novo*, comment s'y prendre pour éviter la catastrophe ? Les conseils suivants me semblent s'imposer :

1. Il faut avoir au sein de l'entreprise de vraies compétences informatiques, c'est-à-dire des ingénieurs capables de comprendre et de critiquer tous les aspects du développement même s'ils ne le font pas eux-mêmes. Ce qui signifie bien entendu que leur activité n'est pas limitée à la rédaction de cahiers des charges et à l'administration de contrats, mais qu'il leur arrive de développer réellement, et pas tous les dix ans. Incidemment, et contrairement à la doxa répandue par les SSII et la presse spécialisée, il n'y a pas de honte à entreprendre des développements avec ses propres personnels.
2. Un bon équilibre doit être trouvé entre les compétences internes et externes. Si les effectifs de développeurs internes descendent en dessous d'un seuil de 40% sur l'ensemble des projets, la situation devient dangereuse. Il s'agit bien sûr ici de vraies compétences en informatique, pas en chefferie de projet ou en comptabilité. Ce ratio de 40% doit être adapté aux circonstances : en situation de croissance d'activité il convient d'augmenter la part des effectifs internes pour ne pas être pris de court devant des besoins nouveaux, par exemple jusqu'à 60%, alors que si l'on prévoit une baisse d'activité il peut être judicieux d'accroître le recours à l'externalisation pour éviter d'avoir plus tard à organiser des plans sociaux. Mais l'idée est de maintenir une situation où l'entreprise garde le contrôle de son SI[2].
3. Il faut aussi trouver un bon équilibre entre maîtrise d'ouvrage et maîtrise d'oeuvre : ce point sera développé à la section suivante.
4. Il faut lire le livre de Brooks

[2] et un livre

sur l'*eXtreme Programming* [3] : même si on ne retient pas toutes leurs idées, il faut les connaître parce qu'elles sont bien meilleures que celles des autres.

5. Après avoir lu les livres conseillés au point 4, il faut mettre en pratique au moins une idée : la spécification du logiciel évoluera et s'affinera au fur et à mesure du développement, il est inutile, voire nuisible, de fixer trop de choses au départ. La rédaction de l'appel à concurrence et le choix d'un éventuel prestataire dépendent fortement de ce principe. Dans un service public français il est très difficile de suivre ce conseil, sauf à faire le développement avec des personnels du service.
6. Le client réel, celui qui sera le bénéficiaire du système à réaliser, doit s'impliquer fortement et concrètement dans le projet. S'il ne le fait pas, il doit renoncer à son droit de parole au profit d'une maîtrise d'ouvrage déléguée qui, elle, devra prendre part activement au développement. Si ni l'une ni l'autre de ces conditions ne peut être remplie, le projet sera un échec, inéluctablement.

Équilibre entre maîtrise d'ouvrage et maîtrise d'oeuvre

Parmi les conditions récurrentes d'échec de projets informatiques figurent les dysfonctionnements de la maîtrise d'ouvrage, de la maîtrise d'oeuvre, des deux, de leurs relations mutuelles, ou la conjonction de plusieurs de ces conditions. Comme énoncé à l'alinéa précédent, le maître d'ouvrage, c'est-à-dire le client réel, doit assumer pleinement ses responsabilités et jouer son rôle, y compris sur le plan technique.

Pendant toute une époque la maîtrise d'ouvrage est restée dans l'ombre parce que l'informatique interne des entreprises, qui jouait le rôle de maître d'oeuvre, campait sur une position de pouvoir inexpugnable, fortifiée par le coût considérable des investissements informatiques. C'était le maître d'oeuvre qui décidait de tout dans les projets de gestion, situation de l'ordre de la catastrophe, parce qu'elle engendrait la frustration des utilisateurs, la rigidité de l'organisation, l'incapacité à conduire des évolutions fonctionnelles ou techniques.

Cette position de force des informaticiens internes a été peu à peu érodée puis finalement ruinée par plusieurs mutations importantes auxquelles ils n'ont en général pas su s'adapter : l'essor de la micro-informatique et de l'informatique des utilisateurs, le développement des progiciels de gestion intégrée, l'externalisation des développements décidée par des directions générales excédées par l'arrogance des informaticiens. Les services informatiques internes ont aussi eu du mal à comprendre l'importance de phénomènes émergents comme l'Internet, les logiciels libres, et les évolutions techniques de manière générale, parce que leur culture informatique était souvent défaillante et leur mobilité intellectuelle limitée.

C'est ainsi que certaines entreprises ont connu une véritable Bérézina de l'informatique interne, qui a ouvert la voie à l'inversion de la situation décrite ci-dessus, au profit de deux autres catastrophes au choix : l'informatique pilotée directement par les utilisateurs, ou l'absorption de la maîtrise d'oeuvre par la maîtrise d'ouvrage. La différence entre les deux cas est que l'informatique abandonnée aux utilisateurs débouche sur l'anarchie et la pagaille totales, alors que, s'il reste une maîtrise d'ouvrage centrale, il est possible de conserver une certaine cohérence au système d'information de l'entreprise. Mais dans les deux cas les développements sont entièrement concédés à des prestataires qui, face à des clients dépourvus de toute compétence technique, ne résistent pas longtemps à la tentation d'en profiter, et il serait difficile de le leur reprocher, dans un monde où chaque entreprise lutte pour sa survie. Les résultats peuvent être mirifiques en termes de budgets. Un consultant indépendant qui préfère rester anonyme m'a confié que les entreprises et les services publics, en France, étaient les principales victimes de ce genre de situation, et que les montants des contrats qui leur étaient proposés étaient souvent triples de ce qui était demandé à un client sérieux pour un travail comparable.

En bref, il faut pour réussir un projet une maîtrise d'ouvrage consistante et décidée à faire son travail, c'est-à-dire à énoncer l'objectif et à l'assumer. Énoncer l'objectif, c'est notamment rédiger des documents « de vision », notes de trois pages au moins et dix pages au plus qui expliquent en langage humain et sans détails superflus ce que l'on veut. Ensuite il faut rédiger les scénarios qui correspondent aux visions, et les jeux de tests qui permettront de les valider. Puis il faut jouer les scénarios avec les équipes de développement, les modifier et recommencer. Pour un gros projet c'est un travail à plein temps, il va sans dire, et qui n'a rien à voir avec des situations que j'ai observées, où le client vient en réunion de projet proférer quelques caprices, de préférence incompatibles avec ceux de la réunion précédente, et retourne à ses occupations extérieures au projet.

Il faut aussi une maîtrise d'oeuvre consistante, c'est-à-dire suffisamment compétente sur le plan technique pour être un interlocuteur critique des prestataires éventuels, et de la maîtrise d'ouvrage. Un informaticien, m'a appris un jour un collègue, c'est quelqu'un qui dit non. Cela ne rend pas forcément populaire, mais nous retrouvons ici la notion de « fonctionnaire wéberien » mentionnée au chapitre 1 par Cornelius Castoriadis ; il serait également possible de remonter plus loin dans le temps, jusqu'à Confucius, qui exigeait du conseiller qu'il donne toujours au roi son meilleur conseil, fût-il désagréable au point de déclencher la colère du souverain et la mise à mort du conseiller.

Relations de travail informatiques

Une fois ces conseils approuvés, il reste à faire le travail. Tout ce que nous avons écrit jusqu'ici concourt à établir l'idée que le travail de développement ne peut être ni spécifié à l'avance avec une précision suffisante, ni encadré par des méthodes tayloriennes. La conséquence qui en découle est que, pour atteindre l'objectif, on ne pourra compter que sur la motivation et la bonne volonté de l'équipe de développement ! Cette conclusion pourra apparaître accablante aux adeptes de la démarche qualité et de la conduite de projet, qui vont m'en vouloir d'avoir au fil des pages critiqué leurs méthodes favorites qui étaient censées éviter une telle extrémité. Mais est-il choquant qu'un employeur ait à compter sur la motivation et la bonne volonté de son personnel ?

Ce que je crois avoir établi au fil de ce livre, c'est que de toute façon la démarche qualité ISO 9000 et la conduite de projet au sens dur *échouent* à contrôler et à canaliser complètement le travail de développement, parce qu'il est dans sa *nature* d'échapper au contrôle et à la canalisation. Et tous les développements réussis que j'ai connus l'ont été parce que les développeurs étaient motivés et de bonne volonté.

Cela dit, tous ceux qui ont essayé de travailler avec des informaticiens savent qu'obtenir ces conditions n'est pas facile, et les maintenir dans le temps, encore moins.

Les informaticiens forment une corporation au sein de laquelle les critères de reconnaissance mutuelle et de hiérarchie reposent sur la compétence3. La corporation est subdivisée en obédiences identifiées par la technique de référence, qui détermine l'axe selon lequel se mesure la compétence : pour les experts de tel langage de programmation, ce sera la virtuosité dans l'écriture de programmes complexes, pour les experts de tel système d'exploitation ce sera la connaissance détaillée du fonctionnement interne de ses parties les plus absconses, etc. Mais tous ont tendance à faire front contre les managers dépourvus de compétence particulière en informatique qui auraient la velléité de les diriger. Et plus ils sont compétents plus ils adoptent cette position frondeuse : à la limite, il faut se méfier des conciliants, ils risquent d'être incompétents.

Face à des représentants de cette corporation informatique, comment faire pour diriger leur activité spontanée vers le but fixé par leur employeur ? Autant dire que le problème n'est pas simple. Nous nous placerons dans le cas d'un travail salarié : le développement de logiciel libre dans un contexte de volontariat et de bénévolat est

assez différent.

Pour qu'une équipe constituée de salariés du maître d'ouvrage ou d'un sous-traitant produise un travail orienté vers un objectif cohérent, nous admettrons qu'il faut un responsable de cet objectif. Le problème à résoudre est celui de la façon dont le responsable va conduire l'équipe vers l'objectif, et donc acquérir une certaine autorité pour ce faire. On sait combien l'autorité est devenue problématique dans notre société : elle l'est encore plus dans le microcosme des informaticiens, qui se voient volontiers comme de petits démiurges face à leur ordinateur et sont très ignorants des conditions sociales et économiques de leur activité â€" ce pourquoi nous avons schématiquement rappelées celles-ci au chapitre 1.

Une première position envisageable est celle du responsable sorti des rangs de la corporation, où il jouissait d'une position élevée sur l'échelle de compétence. Il peut être tenté de rester le plus fort dans son domaine pour asseoir son autorité sur sa compétence. C'est un piège : diriger une équipe prend du temps qui n'est plus consacré à l'activité technique. Les jeunes recrues ont appris en deux ans à l'école ce que les plus vieux ont appris sur le tas, moins bien, en dix ans. L'expérience a des avantages, mais pas pour tous les problèmes, et lutter sur tous les fronts pour être meilleur que chacun de ses collaborateurs dans tous les domaines est impossible. Bref, très vite, pour rester le plus fort le manager va être amené à chasser les bons éléments et à recruter des médiocres ou des pusillanimes. Même ainsi, cette démarche le conduira à l'impasse, parce que pendant qu'il s'épuisera en vain à courir une dizaine de lièvres techniques à la fois, il ne fera pas sérieusement son travail de direction de l'équipe et offrira bientôt l'image désolante du *guru vieillissant*.

Cette critique du responsable qui veut rester supérieur à chaque membre de son équipe dans chaque domaine technique ne doit surtout pas être prise pour une approbation de la seconde position possible : celle du responsable totalement incompétent sur le plan technique. Disons le tout net, un responsable dépourvu de toute compétence et de toute curiosité relatives au métier des membres d'une équipe de développeurs de logiciel ne doit nourrir aucun espoir de la diriger un jour. La position du responsable incompétent est possible au niveau $n+2$, parce que là il a à superviser plusieurs équipes dans des domaines variés et on ne peut pas humainement exiger de lui qu'il soit un spécialiste de chacun, mais au niveau $n+1$ ce n'est pas envisageable.

La bonne position sera donc entre ces deux extrêmes. Je serais tenté de dire que le responsable d'une équipe technique doit être capable d'être au moins un bon second pour chacun des membres de son équipe dans un des domaines de spécialisation de celui-ci : disons que c'est l'idée générale, pas une règle absolue. D'expérience, la plupart des spécialistes ressentent positivement une situation où ils jouent le rôle de mentor vis-à-vis de leur responsable, et où celui-ci reçoit d'eux des compétences qu'il utilisera par la suite.

Le manager doit donc accepter de se présenter face à son équipe en position d'infériorité du point de vue de la compétence technique, au moins sur certains points : chaque membre de l'équipe connaît au moins le domaine dont il s'occupe personnellement mieux que son chef, sinon cela voudrait dire que chaque membre d'une équipe de n personnes ne posséderait qu'une compétence moyenne égale à $1/n \times$(*compétence du chef*), ce qui serait désolant (certains chefs constituent leur équipe ainsi, c'est bon pour leur tranquillité personnelle mais pas pour la qualité de l'équipe). Être le chef de gens plus compétents que vous sur certains points demande un mélange savamment dosé d'humilité et d'autorité et suppose qu'on leur apporte autre chose que de la compétence technique : définition d'objectifs mobilisateurs, capacité à défendre les projets de l'équipe devant les clients, la direction et les autres départements, aptitude à obtenir des financements et à remporter des appels d'offres, création de bonnes conditions de travail pour les membres de l'équipe, talent pour réaliser la synthèse des travaux du groupe. Comme le sait quiconque a essayé, obtenir et conserver l'autorité nécessaire à la direction d'un groupe est le résultat d'une transaction permanente, surtout mais pas uniquement dans les conditions idéologiques contemporaines (on pourra pour s'en convaincre consulter l'autobiographie de Gengis Khan ou les ouvrages qui s'en sont inspirés), et nul dans cet exercice n'est à l'abri de l'usure du temps. Pour mieux comprendre ce sujet délicat on pourra lire avec profit un texte bref mais incisif d'Hannah Arendt [], où elle explique notamment que l'autorité est un moyen d'obtenir l'obéissance qui se distingue à la fois du recours à la contrainte, à la coercition, et de la persuasion par des arguments.

Dans son essai de sociologie de ce milieu un peu particulier[], Philippe Breton a observé des équipes plus ou moins formellement dévolues à l'assistance informatique aux utilisateurs, et il s'est intéressé à l'attitude de ces informaticiens à l'égard de leurs « clients ». La situation semble de prime abord assez différente de celle du responsable d'équipe qui négocie l'établissement de son autorité, mais l'observation de nombreuses occurrences de ces deux types de relations tendrait à suggérer que les leçons de l'une peuvent s'appliquer à l'autre ; après tout il s'agit dans les deux cas d'obtenir qu'un expert veuille bien mettre à votre disposition sa précieuse compétence, dans un rapport de force tel que le recours à la contrainte aboutirait exactement à l'inverse du but poursuivi.

Le résultat obtenu par les observations et analyses de Philippe Breton consiste en ceci que les profanes en quête de conseil et d'assistance auprès des informaticiens se répartissent de façon tranchée en deux catégories : le type A et le type B. Appartiennent au type A les personnes pour les problèmes desquels les informaticiens sont toujours prêts à se mettre en quatre, à passer la nuit s'il le faut, à contacter des collègues à l'autre bout de la planète pour obtenir la variante rare de tel logiciel qui pourrait répondre à une question délicate. Les personnes affublées du type B sont moins bien traitées : leurs appels à l'aide sont toujours mis sous la pile des travaux en attente, leurs problèmes se voient souvent gratifiés, après une longue attente justifiée par une « étude technique » plus ou moins réelle, du diagnostic « incurable », on n'hésite pas à les engager dans de fausses pistes, leurs mésaventures informatiques déclenchent la joie et la bonne humeur des informaticiens, les remarques narquoises ou sarcastiques ne leur sont guère épargnées. Ma longue expérience empirique corrobore tout à fait l'analyse de Philippe Breton. Bref, il vaut mieux se débrouiller pour être du type A si on ne veut pas endurer des moments très désagréables, d'autant plus que l'appartenance à un type semble très stable et durable, très difficile à inverser, et qu'il n'y a pratiquement pas de situation intermédiaire. On imagine comment cette situation peut se transposer aux relations entre un groupe d'experts et leur supérieur hiérarchique (appelé également *n*+1 selon la nouvelle terminologie politiquement correcte).

Quels sont les facteurs qui déterminent l'appartenance à un des types ? Étant donnée la masculinité prononcée des équipes informatiques, on pourrait penser qu'être une femme sachant jouer de son charme suffit à conférer le type A : sans nier l'influence fréquente de ce facteur, cela ne suffit pas à tout coup. Traiter les gens comme des domestiques, les convoquer de façon impérieuse dans votre bureau à l'heure qui vous arrange vous, leur rappeler à haute et forte voix qu'ils sont payés pour vous rendre service, menacer de se plaindre à leur chef ou à la direction et autres comportements du même genre assurent par contre une inscription garantie au type B, et je dois dire que c'est très fréquent. Mais restent beaucoup d'autres exemples plus difficiles à analyser, et pour lesquels Philippe Breton dit qu'il n'existe pas d'explication simple, mais une nébuleuse de facteurs objectifs et subjectifs.

Une fois écartées les questions de courtoisie élémentaire, qui condamnent d'emblée une population importante au type B, il semble que les conditions d'accès au type A tournent autour de deux attitudes :

1. manifester le plus clairement possible que l'on respecte le travail des informaticiens et que l'on ne sous-estime pas ses difficultés ;
2. prouver qu'avant de demander de l'aide on a fait des efforts, dans les limites de ses propres compétences, pour trouver soi-même la réponse à la question par la lecture des documentations et par des essais menés de façon assez systématique.

D'après mes observations personnelles, les attitudes décrites ci-dessus, et notamment l'ampleur des punitions infligées au type B, sont modulées selon que les informaticiens de l'équipe d'assistance ont des emplois de statut stable ou précaire, sont soumis à une domination plus ou moins facile à esquiver. Si l'assistance technique est assurée par une entreprise extérieure qui facture ses interventions, le niveau de courtoisie des demandes qui lui sont adressées est bien meilleur que dans le cas d'une équipe interne, et la satisfaction de toutes les parties est supérieure. Il semble que ce type d'activité soit tout désigné pour l'externalisation : la demande potentielle d'assistance est infinie, si la prestation est gratuite elle ne sera jamais jugée satisfaisante, lui conférer un coût assainit la situation.

Philippe Breton n'a à ma connaissance pas consacré d'analyse aussi systématique aux relations entre les

équipes informatiques et leurs chefs, mais son livre contient beaucoup de notations éclairantes sur ce sujet, et je crois que sa typologie des utilisateurs est pertinente là aussi, moyennant quelques adaptations. Si l'équipe informatique est interne à l'entreprise, le chef de type B aura tout le mal du monde à obtenir quelque résultat, et passer son temps à licencier les gens qui ne font pas leur travail ne fait pas avancer la réalisation. Si l'équipe est externe, toutes les demandes seront bien sûr acceptées avec le sourire et facturées au prix fort, sans que ce soit une garantie de bonne fin, du fait des procédures d'esquive que nous avons déjà évoquées (notamment aux sections 4 et 3).

À quoi les informaticiens passent-ils leur temps ?

Nous l'avons déjà souligné, les informaticiens ont souvent de meilleures relations avec les ordinateurs qu'avec les humains, spécialement lorsque les humains en question sont leurs collègues de travail, et plus particulièrement s'ils sont à quelque titre ceux qui leur demandent un travail. Essayer de combler ce fossé d'incompréhension est une tâche immense à laquelle ce livre s'efforce de contribuer. Nous avons déjà insisté sur l'importance qu'il y avait à reconnaître la difficulté et la complexité du travail informatique, et pour ce faire l'interlocuteur régulier d'informaticiens ne perdra pas son temps en se documentant sur la question par la lecture d'ouvrages tels que celui-ci ou ceux qui figurent dans la bibliographie, ou, mieux encore, en pratiquant lui-même la programmation en amateur, exercice qui lui procurera, outre un certain plaisir, une compréhension accrue de toutes sortes de questions qui surgissent dans les conversations avec ses interlocuteurs techniques.

Un trait de comportement des informaticiens qui suscite souvent l'incompréhension de leurs interlocuteurs, c'est leur emploi du temps. Il est généralement impossible à un profane d'estimer la complexité d'un travail informatique quelconque, qu'il s'agisse de maintenance, de paramétrage ou surtout de développement, et partant le temps nécessaire à son accomplissement. La règle est que le profane sous-estime les délais, et ne comprend pas que le travail demandé ne soit pas réalisé immédiatement. À une certaine époque l'informatique a fait irruption dans la recherche en biologie moléculaire sous la forme de l'*analyse de séquences*4, une activité au demeurant mal définie, mais ce qui est sûr c'est qu'au début des années 1990 le chercheur moyen pensait qu'il lui suffisait de donner la disquette où était enregistrée sa séquence à un bioinformaticien pour avoir le résultat vingt minutes plus tard, alors que le travail réel demandait plutôt trois semaines, et surtout une discussion approfondie sur le type d'analyse attendue. Cette divergence de vue conduisait droit au malentendu. Nul doute que si le chercheur s'était un peu donné la peine de voir lui-même de quoi il retournait, il aurait mieux compris pourquoi une telle attitude le condamnait à un classement direct en type B, et comment l'éviter.

Une autre chose semble difficile à comprendre : la maîtrise d'outils complexes, tels que système d'exploitation, environnement de développement, langage, compilateur, éditeur, compte pour beaucoup dans la compétence d'un informaticien. Il est donc légitime qu'il consacre beaucoup de temps à se perfectionner dans leur usage, à se tenir au courant de leur évolution et de l'apparition de nouveautés, à en parler avec des collègues. D'où la question que l'on devine souvent au bord des lèvres de certains interlocuteurs : « Mais qu'est-ce qu'il fabrique au lieu de s'occuper de mon problème ? ». De même que pour construire un mur il faut d'abord construire un bon échafaudage, la construction d'un système informatique demande la préparation d'outils adéquats, et éventuellement leur apprentissage, ce qui prend du temps. Le travail du développeur consiste pour une part à être à l'aise en permanence au milieu de ses outils.

Laconiques leçons de l'expérience

J'ai posé à un certain nombre de responsables d'équipes de développeurs, certaines de grande dimension, la question suivante : comment obtient-on que les gens travaillent, et non seulement qu'ils travaillent, mais que le résultat soit ce qui leur était demandé au départ ? Les réponses, dont nous donnerons le résumé ci-dessous, sont assez convergentes.

1. Avant toute chose, il faut être très attentif lors du recrutement. Le candidat doit rencontrer au moins une personne très compétente et expérimentée dans son domaine technique, qui saura détecter s'il possède réellement les compétences qu'il prétend avoir et s'il est capable de les mettre en oeuvre : l'informatique est en effet un domaine où l'exhibition d'un vocabulaire à la mode fait souvent illusion. Il faut aussi évaluer la personnalité du candidat, et vérifier pendant la période d'essai son aptitude à travailler avec l'équipe à laquelle il est destiné. Les *gurus* géniaux mais irréductiblement individualistes ne sont pas forcément à leur place dans une équipe en entreprise. Curieusement, beaucoup de candidats à un recrutement avouent finalement assez ouvertement leur peu de goût pour le travail : autant amener ce sujet dans la conversation et écouter les réactions.
2. Pour recruter des spécialistes techniquement compétents, tels que développeurs ou ingénieurs systèmes et réseau, il faut éviter de diluer le message et de se tromper de cible. Il faut d'abord écarter tout risque de confusion avec les populations des pseudo-experts en génie logiciel déjà signalés à la section 2 et des aspirants chefs de projet : sachez que si les deux premières lignes de votre annonce comportent des locutions telles que « schéma directeur », « conduite de projet », « démarche qualité », tous les vrais experts techniques seront déjà passés à l'annonce suivante, parce qu'ils savent que dans un tel contexte ils vont passer leurs journées en réunions mortelles au lieu de faire ce qui les intéresse : construire des systèmes informatiques. La mention d'une compétence relative à un logiciel d'application tel que SAP pourra attirer des candidats versés dans les systèmes financiers, mais écartera à coup sûr les vrais développeurs. Pour attirer un vrai développeur, il faut être un tant soit peu de la partie, parler son langage.
3. Une fois l'équipe constituée, il faut entretenir son élan et sa motivation. Tous les avis convergent pour dire qu'un rôle très important est joué par la conscience d'une communauté de destin entre chaque membre de l'équipe et l'entreprise : si le projet échoue l'entreprise peut disparaître, ou du moins l'équipe. Cette conscience est évidemment plus facile à susciter dans une petite entreprise que dans la fonction publique. Le rôle d'entraînement des responsables et la confiance qu'ils inspirent sont aussi très importants.

Voilà, il n'y a pas de recette miraculeuse, uniquement des connaissances que les meneurs des groupes de chasseurs du paléolithique supérieur possédaient déjà. Ce qui ne veut pas dire que tout le monde les possède, ni que ce soit facile de les mettre en application.

1Je propose la traduction *appliquette*, ou *applète*. 2Résumé d'une communication de Jean-Marie Faure au Club des Maîtres d'Ouvrage. 3Il faudrait mentionner ici le clivage entre le milieu des chercheurs patentés et celui des ingénieurs et techniciens, qui communiquent peu entre eux et se vouent une certaine méfiance mutuelle d'où le mépris n'est pas toujours absent, mais nous parlerons peu ici des chercheurs. 4Les séquences biologiques décrivent des macro-molécules de trois sortes, les protéines, les molécules d'ADN et les molécules d'ARN. Chacune de ces molécules est constituée par l'assemblage en séquence d'éléments de base pris parmi les acides aminés, au nombre de 20, pour les protéines, ou parmi les acides nucléiques, au nombre de 4, pour l'ADN et l'ARN. La succession de ces éléments, considérés comme les caractères d'un alphabet, peut être vue comme un texte, étonnamment bien conservé par l'évolution, dont l'analyse est au coeur de la biologie moléculaire contemporaine[].

Page d'accueil de ce livre

---

[1] Laurent Bloch et alii. *Rapport d'activité du Service d'Informatique Scientifique*. Institut Pasteur, 17 novembre 1998. http://www.pasteur.fr/recherche/uni....

[2] Frederick P. Brooks, Jr. *The Mythical Man-Month* (traduction : *Le mythe de l'homme-mois*). Addison-Wesley (Vuibert pour la traduction),

Reading, Massachusetts (Paris), 1975-1995.

[3] Jean-Louis Bénard, Laurent Bossavit, Régis Médina, Dominic Williams. *L'eXtreme Programming.* Eyrolles, Paris, 2002.