La pensée aux prises avec l'informatique

Systèmes d'information

Laurent Bloch
25 août 2024

Du même auteur

Initiation à la programmation et aux algorithmes avec Scheme, Technip éd., 2020

Initiation à la programmation et aux algorithmes avec Python, Technip éd., 2020

Révolution cyberindustrielle en France, Economica éd., 2015

Sécurité informatique – Pour les DSI, RSSI et administrateurs, Eyrolles éd., 5ème édition, 2016

L'Internet, vecteur de puissance des États-Unis ? - Géopolitique du cyberespace, nouvel espace stratégique, Diploweb éd., 2017

Splendeurs et servitudes des systèmes d'exploitation – Histoire, fonctionnement, enjeux, Laurent Bloch éd., 2020

Site Web de l'auteur

(comporte des documents complémentaires)

https://laurentbloch.net/MySpip3/-Systemes-d-information-

Table des matières

Préface
de Michel Volle

1

> *La différence entre fonction et procédure est un reflet de la différence plus générale entre décrire les choses et décrire la façon de faire les choses [...]. En mathématiques, nous utilisons des descriptions déclaratives (qu'est-ce que c'est ?), alors qu'en informatique nous utilisons des descriptions impératives (comment faire ?).*
> (Harold Abelson et Gerald Jay Sussman)

Il est salubre, il est nécessaire de philosopher sur l'informatique.

Abelson et Sussman [1] disent qu'elle est concernée par le savoir-faire, le « *how to* », alors que les mathématiques seraient concernées par les définitions, le « *what is* ». Cette phrase est à la fois profonde et inexacte.

Elle est inexacte parce que les mathématiques qu'ils évoquent ne sont pas celles des chercheurs mais celles des lycées, des grandes écoles et de l'agrégation qui se limitent à utiliser, pour traiter des « problèmes » qui ne sont au fond que des questions de cours, les outils mis au point par les chercheurs d'autrefois.

Elle est profonde parce que la *pensée explicite* que l'on nous a enseignée, dressée à procéder de façon déductive à partir d'axiomes dont on ne questionne pas la pertinence, s'appuie en effet sur des définitions. Le savoir-faire, l'aptitude à résoudre les problèmes pratiques, sont laissés à la *pensée implicite*, intuitive, quotidienne, plus naturelle sans doute mais moins formelle et moins « rigoureuse » que la pensée explicite.

L'informatique, en invitant à expliciter la pratique, à articuler la pensée et l'action, brise le dogmatisme d'un enseignement qui a oublié la démarche expérimentale pour ne transmettre que les résultats de l'expérience.

*

Mais on peut interpréter autrement la phrase d'Abelson et Sussman. Considérons non plus le cadavre que l'on présente dans les amphis, mais les mathématiques vivantes, celles qui explorent la part du *monde de la pensée* qui est soumise au principe de non-contradiction[2]. Le chercheur en mathématiques possède non seulement des définitions, mais aussi le savoir-faire qui lui permet de mettre à jour des systèmes d'axiomes *féconds*. Ce savoir-faire suppose qu'il puisse, selon une intuition qui enjambe les étapes du raisonnement, anticiper des résultats non encore établis.

La frontière entre l'informatique et les mathématiques ne passe donc pas entre *définition* et *savoir-faire*, mais entre un savoir-faire adapté au monde de la pensée, seul objet des

1 Michel Volle nous a quittés en 2024. Il a été le conseiller du présidents d'Air France (Christian Blanc) et du DG de l'ANPE (Michel Bernard). Il a créé deux entreprises, son blog est ici : `https://michelvolle.blogspot.com/`.

2 Elle ne recouvre pas *tout* le monde de la pensée : le théorème de Gödel (1931) montre qu'il existe, quel que soit le système d'axiomes que l'on utilise, des propositions évidemment vraies et cependant indémontrables.

mathématiques, et le savoir-faire qui convient face au *monde de la nature* (en entendant par « monde de la nature » tout ce qui se présente comme obstacle ou comme outil devant notre aspiration au bien-être : la nature physique bien sûr, mais aussi la nature humaine et sociale).

Ce n'est pas pour penser en effet que nous utilisons l'automate, mais pour *agir*. Pour penser, nous disposons de notre cerveau, des associations d'idées qu'il suggère sans arrêt et des raisonnements auxquels nous le contraignons. C'est sur le cerveau, et non sur l'automate, qu'il faut compter pour exercer le « simple » bon sens[3]. L'automate, lui, *assiste notre action* en offrant sa capacité à classer, trier, traiter, transformer, transporter les données que fournit l'observation.

L'informatique a ainsi une finalité essentiellement *pratique*. Relativement récente, elle pose à l'horizon de l'intellect des questions dont la nouveauté dérange l'inertie des corporations. D'où, de la part de ceux qu'elle aurait dû intéresser le plus, un mépris qui confine à la fureur. C'est parce que l'informatique pose des questions philosophiques nouvelles que les philosophes, qui pour la plupart préfèrent méditer les grands textes de leur discipline plutôt que de se confronter au monde de la nature, produisent des considérations tantôt apocalyptiques, tantôt enthousiastes, presque jamais pondérées ni exactes[4]. C'est parce qu'elle risquait de déranger l'échiquier des pouvoirs universitaires que les mathématiciens l'ont empêchée de prendre, dans l'enseignement, la place qui doit lui revenir.

*

On savait depuis longtemps construire des automates. On apprit à les programmer au XIX^{e} siècle. Mais le canard de Vaucanson, comme le métier à tisser de Jacquard, sont spécialisés dans une seule fonction. Il fallut un admirable effort d'abstraction pour concevoir l'automate *essentiellement* programmable, l'automate capable de commander l'exécution d'un programme à *n'importe quel type* de périphérique (écran-clavier, bras articulé d'un robot, avion en pilotage automatique, centrale nucléaire etc.)

L'informatique est née avec l'entreprise moderne[5] – forme spécifique d'organisation du travail humain, du rapport à la nature et du rapport au marché – dans le *loop* de Chicago à la fin du XIX^{e} siècle. Elle s'est développée parallèlement à d'autres outils – classeur mécanique, trombone, machine à écrire, photocopieuse, post-it, téléphone etc. – qu'elle a ensuite partiellement absorbés. À l'*être humain organisé* en un réseau de compétences complémentaires (EHO) elle a articulé l'*automate programmable doué d'ubiquité* (APU), plate-forme de mémoires, processeurs, réseaux et logiciels. Les processus de production en ont été transformés ainsi que les conditions de l'échange (distribution des produits, traitement des effets de commerce).

L'EHO et l'APU s'entrelacent jusque dans le détail de chaque processus de l'entreprise selon un « modèle en couches » qui se divise en couches plus fines. Ce type de modèle est l'une des innovations philosophiques qu'a apportées l'informatique : il permet de représenter les situations où plusieurs logiques se conditionnent mutuellement, situations fréquentes dans la vie courante mais que seul un modèle en couches permet de *penser*.

Pour s'informatiser, l'entreprise doit expliciter ses règles de gestion, le flux de ses processus, son référentiel : elle doit se *modéliser*. Cela implique qu'elle choisisse les êtres qu'elle

3 « Ces machines sont dépourvues de bon sens : elles font exactement ce qu'on leur a demandé de faire, ni plus ni moins. C'est ce qui est le plus difficile à comprendre quand on essaie d'utiliser un ordinateur pour la première fois. » ([63] vol. I p. v.)

4 François Jullien a construit une passerelle solide entre la pensée chinoise et la pensée grecque. On attend le philosophe qui construira une passerelle solide entre la philosophie et l'informatique.

5 Nous désignons par le terme « entreprise » toutes les institutions qui produisent des biens et des services, qu'il s'agisse d'entreprises au sens juridique du terme ou de services publics producteurs d'externalités positives (y compris donc l'armée, la justice etc.)

représentera, les variables qu'elle observera sur ces êtres et les traitements qu'elle leur appliquera ; dans le langage des informaticiens, qui abonde en faux amis, on dira que l'entreprise doit définir ses « objets » avec leurs attributs, opérations, règles de gestion, cycles de vie etc. Cela suppose qu'elle sache, dans la complexité sans limites du monde de la nature, découper les objets, attributs etc. *pertinents* pour son action en faisant abstraction des autres. L'informatique assouplit l'abstraction pour servir une visée opérationnelle. Cette *pratique de l'abstraction*, qui isole du reste du monde les êtres qu'implique l'action, chacun la suit dans la vie courante mais elle est rarement explicite.

C'est, par rapport à la pensée socratique, un renversement de perspective : le concept ne révèle plus l'essence des choses mais nous outille pour représenter les êtres concernés par l'action. Le processus, comme un tourbillon dans un fleuve, brasse une matière sans cesse renouvelée et l'objet qui le parcourt (dossier d'un client, d'un produit, d'un salarié etc.) conserve son identité tout en se transformant. Cela nous place plus près de Lao Zi (VI^e^ siècle avant JC) que de Parménide (540-450).

En construisant le référentiel qui définit les objets, attributs et codages, l'informatique définit le *langage* de l'entreprise. Seul sera dicible et audible le recours aux identifiants et nomenclatures qu'elle met en œuvre. On ne pourra pas, dans l'entreprise, utiliser une autre segmentation des clients, un autre référentiel de l'organisation, une autre classification des produits que ceux qui sont incorporés à ses logiciels. Celui qui voudrait modifier le langage – et on a souvent de bonnes raison de vouloir le faire – devra passer par une modification du référentiel. Ainsi l'informatique exige que l'on explicite l'évolution du langage, cette évolution que les structuralistes avaient dédaignée pour mieux nous serrer dans le corset du langage à l'œuvre.

L'automate traduit, selon une cascade de langages, la plus simple des manœuvres de l'utilisateur en des milliers d'instructions élémentaires exécutées en une fraction de seconde. Pour qu'il puisse y parvenir ses concepteurs ont tiré parti des propriétés électroniques de la matière : cet artefact appartient donc au monde de la nature physique. Ses ressources – mémoire, puissance, débit du réseau – sont bornées par leur dimensionnement. L'informaticien est ainsi un physicien qui traite avec précautions (qualité, sécurité) une masse (de données) et des vitesses (de traitement).

La programmation est l'exercice intellectuel le plus sain qui soit. Elle ne relève pas, comme on le dit avec condescendance, des mathématiques appliquées. Elle ne peut être réussie que si l'on a explicité le problème à traiter ainsi que tous ses cas particuliers : cela donne parfois au mathématicien l'occasion de découvrir des cas que la simplicité de la « formule » générale lui avait masqués. Le programmeur doit par ailleurs ruser avec la physique de l'automate, et cela le contraint à un *réalisme* dont l'acquisition demande un long apprentissage.

*

Articulation des logiques, évolution du langage, élucidation de la dynamique des processus, pratique de l'abstraction, réalisme : l'informatique apporte du grain à moudre au penseur. En retour l'intervention de celui-ci est nécessaire car, si l'informatique est née dans et pour l'entreprise, si elle a été sécrétée par l'entreprise, il s'en faut de beaucoup que l'entreprise la comprenne. Elle inspire aux dirigeants, tout comme aux philosophes, des sentiments qui oscillent de l'horreur à la fascination. Nous avons donc grand besoin d'explications claires, d'une réflexion en bon ordre, d'une saine pédagogie pour faciliter son appropriation par l'entreprise ou, comme on dit, son « alignement stratégique ».

Les médias, les films, donnent de l'informatique une image faussée : l'informaticien y apparaît comme un « génie » qui tape du code en champion de dactylographie ; les ordinateurs y parlent, pensent et souffrent comme vous et moi. La commodité des interfaces a par ailleurs répandu l'idée que l'informatique était facile : il est donc inutile de se donner

la peine de la comprendre ! Les étudiants se détournent de la programmation, qu'ils jugent trop aride et que l'on préférera sous-traiter à des pays où les salaires sont bas. Pourtant elle fait partie des savoirs fondamentaux sans lesquels on ne peut pas comprendre, aujourd'hui, notre rapport avec le monde de la nature.

Alors que l'articulation de l'EHO et de l'APU, la délimitation de ce qu'il faut ou ne faut pas automatiser, posent une question philosophique et pratique des plus importantes [6], l'attention du grand public, des dirigeants et des penseurs est ainsi accaparée par des images fantastiques. Les informaticiens, sur la défensive, s'enferment dans leur spécialité et se protègent par leur jargon, à moins qu'ils ne s'emploient à faire de la propagande sans mentionner les obstacles[7].

*

C'est à l'exploration des obstacles que Laurent Bloch s'est attelé : obstacles physiques avec lesquels on doit ruser, obstacles sociologiques et intellectuels aussi. La racine de ces derniers est souvent philosophique ou même métaphysique, l'enjeu étant de *savoir comment penser le monde pour pouvoir agir sur lui.*

Laurent Bloch est de ces esprits curieux qui étudient et réfléchissent sans relâche. Il se donne de surcroît la peine de mettre en ordre le résultat de ses travaux, de les exprimer le plus clairement possible pour aider les autres à progresser[8]. Il y faut de la générosité, du dévouement : beaucoup de lecteurs supposent que si un texte est facile à lire, c'est qu'il n'était pas difficile à écrire. Son ouverture d'esprit l'a conduit à s'intéresser de près aux *utilisateurs* de l'informatique : il fait, avec une patience inlassable, bénéficier le club des maîtres d'ouvrage des systèmes d'information[9] d'une pédagogie bienveillante.

Laurent Bloch explicite ici, sans jargon ni complaisance, les apports et limites des méthodes (Merise, UML, SADT, XP etc.) Il présente les diverses couches de l'informatique pour bien faire apparaître le jeu de leurs interactions. Il illustre son propos par des exemples. Son expérience lui permet d'évoquer des obstacles de toutes natures et d'indiquer comment les contourner. Il évoque avec humour les obstacles sociologiques, si irritants : mieux vaut, en effet, les prendre avec patience, avec le sourire !

6 « "Que faut-il automatiser ?" est pour la civilisation d'aujourd'hui l'une des questions les plus suggestives au plan pratique comme au plan philosophique » ([41], p. 92.) »

7 C'est ce qu'a fait Bill Gates (cf. [45]).

8 J'ai étudié avec profit *Initiation à la programmation avec Scheme*[13] et *Les systèmes d'exploitation des ordinateurs*[14].

9 `http://www.clubmoa.asso.fr/`

Avant-propos

L'observation du fonctionnement et des effets de l'informatique dans les secteurs de l'activité humaine qu'elle affecte (c'est-à-dire à peu près tous) révèle des succès et des échecs qui ne sont pas répartis au hasard. L'Internet, le logiciel de pilotage des Airbus, celui des téléphones portables sont des succès considérables. Les Systèmes d'Information de gestion des entreprises sont en revanche souvent des échecs ou du moins des déceptions, cela malgré une complexité technique moindre et un domaine mieux connu, en apparence du moins. Ce livre se propose d'avancer quelques hypothèses sur l'origine de ces disparités.

Dès que l'informatique fut employée à la gestion des entreprises se posa la question du coût des ordinateurs (dont il ne sera pas question ici) et de leur programmation, sujet traité dans ce livre. La création d'un logiciel se révéla difficile, longue, chère, pour un résultat incertain.

Les premières tentatives de maîtrise des coûts furent tayloriennes et échouèrent. Le modèle d'organisation en vigueur depuis une trentaine d'années est le « mode projet » ; s'il a incontestablement permis aux sociétés de services d'améliorer la gestion de leur personnel et de limiter les risques de contentieux avec leurs clients, il n'a pas accru de façon significative le taux de satisfaction de ces derniers : ce taux plafonne en dessous de 40%, valeur que n'accepteraient aucune industrie ni aucun service. La « démarche qualité », censée consolider l'ensemble, aboutit souvent à la bureaucratie et finalement grève les budgets sans bénéfice réel.

Sera développée ici l'hypothèse que cet insuccès fréquent procède d'une mauvaise compréhension de la nature même du travail de réalisation d'un logiciel. L'échec survient lorsque la dimension intellectuelle de ce travail n'est pas reconnue et que la difficulté relative de ses différentes composantes est mal évaluée. La thèse centrale de ce livre énonce que la programmation n'est pas une fabrication, mais un acte de conception, et des plus complexes : c'est cet acte qu'il s'agit de comprendre et d'accomplir.

Cela pose la question du rapport paradoxal entre travail et pensée : pour accomplir un travail qui comporte une part de conception, il faut penser, mais un travailleur expérimenté peut (et même doit) accomplir certaines phases de son travail sans avoir à y penser, ce qui lui permet d'être rapide. Le logicien anglais Alfred North Whitehead a même pu écrire, dans une perspective plus générale, « *civilisation advances by extending the number of important operations which we can perform without thinking about them* » (« la civilisation progesse dans la mesure où le nombre de choses importantes que nous pouvons accomplir sans avoir à y penser augmente »).

La question est donc, ici comme pour d'autres travaux de conception, de savoir penser aux bonnes questions aux bons moments. Les approches bureaucratiques ignorantes de la nature du travail cherchent à éliminer toute pensée, activité coûteuse dont la rentabilité n'est pas immédiatement perceptible. D'où l'échec, que le « perfectionnement » des procédures ne fera qu'amplifier. Plusieurs approches passées et présentes seront examinées : nous verrons qu'ici comme ailleurs l'enfer est parfois pavé des meilleures intentions. L'application trop systématique d'idées parfaitement logiques peut engendrer des catastrophes.

L'examen de ces questions nous amènera à évoquer brièvement les conditions préalables qui s'imposent à toute pensée, et donc à tout travail : à l'heure où la réflexion sur les usages sociaux de l'informatique hésite entre le Charybde de la techno-science obscurantiste et le

Scylla d'une nouvelle barbarie qui ne veut rien savoir des techniques qu'elle utilise, seule une véritable approche multi-culturelle peut nous aider à voir clair et à agir juste.

*

Les dernières années du siècle passé ont vu les cercles dirigeants des entreprises, des administrations et des collectivités publiques, en tout cas en France, détourner leur intérêt des projets informatiques pour le concentrer sur la construction du système d'information de l'entreprise. Cette nouvelle orientation correspondrait, dit-on, à un progrès dans l'assimilation des nouvelles technologies : les problèmes techniques de l'informatisation, triviaux, seraient enfin résolus, et les managers[1], au sein d'une organisation « recentrée sur son métier de base », pourraient prendre du recul et de la hauteur pour se consacrer au recueil et à la canalisation des flux d'information destinés à converger vers le tableau de bord de l'entreprise, grâce auquel le groupe dirigeant pourrait prendre en toute connaissance de cause des décisions stratégiques pertinentes.

Pendant plus de quarante ans l'auteur de ces lignes a participé ou assisté à de nombreux projets de création de systèmes d'information aux objectifs divers et de styles variés, avec des fortunes également diverses et variées[2]. Il appartient à plusieurs associations professionnelles et autres types de cénacles, et participe à d'innombrables colloques et conférences, tous consacrés, sous les angles les plus inattendus ou les plus convenus, à la recherche de la meilleure façon de construire tel ou tel type de système d'information. Il a, bien sûr, lu des milliers de pages et participé à des centaines d'heures de conversation consacrées à ces sujets.

Cette participation assidue ne lui a pas apporté la conviction que les problèmes techniques de l'informatisation fussent résolus, même s'ils ont changé d'aspect. L'ambition panoptique du système d'information intégré lui semble pour le moins questionnable, et le présent ouvrage posera cette question. Quant à la phraséologie du recentrage sur le métier de base, que l'on pourrait traduire par « persévérer dans la routine et mettre tous ses œufs dans le même panier », nous essaierons de savoir qui la produit et à qui elle profite.

Par ailleurs, le monde a vu la construction de réalisations informatiques grandioses telles que l'Internet et tous les systèmes informatiques embarqués qui animent désormais les appareils de notre vie quotidienne, de la console de jeux à l'automobile. Il faudra aussi déterminer les différences, éventuellement explicatives, entre ces projets couronnés de succès et ceux des alinéas précédents, voués à un échec apparemment inéluctable.

Trois classes de systèmes informatiques

Avant d'entamer ce livre il convient de donner une précision : les systèmes informatiques sont envisagés classiquement comme répartis en deux classes, ceux qui sont destinés à faire fonctionner les systèmes d'information des organisations, que nous regrouperons dans la classe 1, et ceux qui assurent la commande et le contrôle des systèmes techniques

1 Que le lecteur se rassure, l'auteur de ce livre fuit les anglicismes, mais veut rétablir dans la langue française de vieux mots qui ont connu meilleure fortune outre-Manche ou outre-Atlantique, comme manager (qui n'est autre que « ménager », et qu'il convient de prononcer « ma-na-jé ») ou challenge.

2 Je suis entré en informatique en 1968. Cette époque était un peu une charnière, pour l'informatique en tout cas. Les vrais intellectuels tel Marcel-Paul Schützenberger en avaient épuisé les charmes, du moins ceux qui étaient dignes de leur intérêt. Les petits arrivistes du management n'avaient pas encore compris que l'on pouvait y gagner du pouvoir et de l'argent. Bref, le milieu était calme, l'intérêt modéré, les formations spécialisées et les diplômes rares et peu connus : pour s'engager dans cette voie il suffisait en fait d'en manifester le désir, ce que j'ai fait, d'abord dans les administrations économiques et financières (Insee, Commission de Développement de l'Informatique du Ministère de l'Économie et des Finances), puis dans quelques institutions publiques ou privées dévolues à la recherche scientifique (Institut National d'Études Démographiques, Conservatoire National des Arts et Métiers, Institut Pasteur, Institut National de la Santé et de la Recherche Médicale, Université Paris-Dauphine).

(trains, avions, téléphones, etc.) rangés dans la classe 2. Il faut à notre avis créer une classe 3 qui regroupe les systèmes destinés à faire fonctionner l'informatique elle-même : systèmes d'exploitation, systèmes de développement, compilateurs, systèmes de gestion de configuration et logiciels de réseau. Ces trois classes de systèmes sont dotées chacune de leur culture technique propre, et les personnes qui travaillent dans chacun de ces domaines sont issues de formations différentes, ont des habitudes de travail différentes et répondent à des analyses sociologiques différentes. Le présent ouvrage est essentiellement consacré aux systèmes de la classe 1, et il n'y sera question de ceux des autres classes qu'incidemment, pour mettre en lumière ce qui les distingue de la classe 1, et pourquoi. Nous pouvons annoncer dès cet avant-propos que les systèmes de classe 1 sont ceux qui rencontrent le plus souvent l'insuccès, bien qu'ils soient les moins complexes sur le plan technique. Il semble bien que les difficultés rencontrées par leurs maîtres d'ouvrage et par leurs maîtres d'œuvre soient d'ordre sociologique plus qu'informatique, et c'est ce qu'il nous faudra examiner.

Il existe des logiciels qui n'entrent dans aucune de nos trois classes, comme par exemple les logiciels généraux (traitement de texte, progiciels de bases de données...) ou les logiciels scientifiques développés par des chercheurs pour leurs besoins propres : ce sont des logiciels autonomes, qui ne constituent pas des *systèmes*, même s'ils peuvent en être une partie ou un composant. Un système se caractérise par l'intégration de plusieurs composants complexes[3] [80].

Le poids des obstacles

Le projet de se doter d'un système d'information, même si l'on en admet la légitimité, est fréquemment l'occasion de malentendus et d'erreurs de méthode qui engendrent retards et dépenses imprévues, quand ce ne sont pas échecs et frustrations. En fait, les études classiques menées sur ce sujet de façon récurrente par de grands cabinets internationaux tels que le Standish Group [105, 106] ou le Gartner Group [90] égrènent la litanie de taux d'échec ahurissants (seuls 34 % des projets atteindraient les objectifs fixés), et ce sans que l'écoulement du temps, scandé par l'apparition de nouvelles méthodologies et de nouveaux principes organisationnels jaillis de l'imagination fertile des cabinets de conseil et des universitaires, semble apporter un progrès qui soit à la mesure des promesses. Ces études sont internationales, mais les projets français ne s'y distinguent pas par l'excellence, loin de là. L'édition 2003 du rapport Standish porte sur treize mille cinq cents projets ; 15 % sont purement et simplement des échecs, les dépassements budgétaires sont en moyenne de 43 %. La liste des causes d'échec est une véritable rengaine : devis incomplets, mauvaise participation des utilisateurs, cahiers des charges délirants établis par la maîtrise d'ouvrage, modifications intempestives des spécifications...

Aucun auteur ou propagateur d'une méthodologie ancienne ou nouvelle ne manque de faire état de ces chiffres catastrophiques pour inciter son lecteur ou son auditeur à s'engager dans la voie supposée salvatrice dont il dépeint les cheminements lumineux.

L'idée qui conduit ici la plume de l'auteur est que ces échecs dans l'édification d'un système d'information destiné à gérer l'entreprise sont prévisibles et destinés à se reproduire. C'est le succès qui serait un accident. D'ailleurs beaucoup de succès proclamés sont en fait moins éclatants si l'on va y voir de près. Qui souhaite proclamer qu'il a échoué ? Tout pousse à l'embellissement des résultats.

Pourquoi cette persistance dans l'insuccès, depuis si longtemps ? Parce qu'un système d'information, aussi « stratégique » soit-il, n'est jamais qu'une combinaison de systèmes informatiques, et que la construction sûre de systèmes informatiques robustes est toujours une entreprise incertaine. Parmi les nombreuses plaisanteries professionnelles des informaticiens, l'une énonce que si les architectes et les urbanistes construisaient les villes comme

3 Les numéros entre crochets, tels qu'ici [80], renvoient à la bibliographie en fin de volume.

les informaticiens construisent les logiciels, un pivert pourrait détruire la civilisation. On ne saurait mieux dire. Il n'empêche, aujourd'hui la civilisation incorpore beaucoup de systèmes informatiques, et pour être juste il faut dire que beaucoup fonctionnent finalement assez correctement, ceux que nous évoquions quelques lignes plus haut par exemple : l'Internet, les jeux électroniques, le système de pilotage de la ligne 14 du métro parisien… Pourquoi réussit-on mieux à faire fonctionner des jeux électroniques que des logiciels de gestion comptable ? Pourtant la programmation de jeux est bien plus complexe et mobilise des connaissances scientifiques bien plus rares que celle des applications de gestion. Nous aurons bien sûr à examiner cette question.

Pourquoi en sommes-nous là, près de soixante ans après l'invention de l'ordinateur ? Parce que la nature du travail de construction de logiciels informatiques n'est pas correctement comprise, et du coup l'organisation de ce travail est, très souvent, conçue selon des plans erronés. La mauvaise compréhension de la nature du travail conduit à le confier à des personnes qui ne sont pas forcément les plus appropriées, qui ne reçoivent pas forcément la formation souhaitable, dont les objectifs ne sont pas fixés selon des critères pertinents, etc.

Pour essayer de comprendre cette situation d'échec dans des projets pourtant abordés avec des motivations fortes et des moyens parfois pharaoniques, nous essaierons de mieux comprendre la nature du travail en général, et du travail de réalisation de systèmes informatiques en particulier. En proposant certains éléments d'explication de ces échecs, nous mettrons au jour ce qu'il faut bien appeler des facteurs de sous-développement au sein de nos sociétés qui se perçoivent pourtant plutôt comme développées.

*

La liste de tous ceux à qui ce livre doit quelque chose serait trop longue pour que je prenne le risque, en la dressant, d'en oublier trop. Je citerai seulement Dominique Sabrier, pour ses relectures toujours précises et d'une exigence judicieuse, et mes collègues de l'Inserm, dont le travail a été pour moi un objet constant de réflexion. Merci à Marc Jammet, des Éditions Vuibert, pour son ouverture d'esprit et sa confiance renouvelée. Le Club des Maîtres d'Ouvrage [31], sous le magistère éclairé de Michel Volle et la présidence de Jean-Marie Faure, a été pour moi une source d'inspiration d'une vivacité incomparable. Les passages de ce livre qui doivent à Michel Volle un conseil, un éclaircissement ou une contribution sont trop nombreux pour que je puisse les indiquer tous. Françoise Courivaud, Hakim Saad, Michel Gaudet, Manuel Serrano, Benoît Picaud, Isabelle Perseil, Juliette Poupard et Marcel Moiroud ont relu, utilement commenté, conseillé et encouragé. Mon ami et collègue William Saurin a procuré les objections acerbes qu'il convenait de relever, et apparaît secrètement comme un personnage de ce livre. Mon ancien collègue Jean-Jacques Kasparian a exercé une influence à longue distance, et apparaît lui aussi comme personnage. La responsabilité des erreurs qui subsistent néanmoins dans ce texte ne peut être imputée qu'à l'auteur. Ce livre a été écrit, composé et mis en page au moyen de logiciels libres, notamment Linux, GNU/Emacs, TEX et LATEX : il convient d'en remercier ici les auteurs et contributeurs, dont le travail désintéressé élargit le champ de la liberté d'expression.

Une grande partie de ma vie professionnelle s'est déroulée au sein du service public ou dans sa proximité. Beaucoup d'exemples de ce livre sont donc pris dans la pratique des administrations et des établissements de recherche. Mes critiques à l'égard de leur fonctionnement administratif ne sont pas faites dans un esprit polémique, leur but unique est d'éclairer des écueils pour permettre, peut-être, de les éviter. Les concepteurs et constructeurs de systèmes d'information pourront aussi trouver que je n'ai guère d'indulgence pour leur travail : bien au contraire, c'est du spectacle de leur entreprise pleine d'espoir face à des obstacles considérables qu'est né ce livre, qui est un essai pour indiquer quelques chausse-trappes et mauvaises pistes à éviter.

*

L'*italique* est utilisé pour la définition d'un terme caractéristique du sujet traité, ou pour un mot étranger, ou pour un nom de marque.

Les termes `en style machine à écrire` sont des mots de langages informatiques (il n'y en a que très peu, à titre d'exemples).

Des matériaux complémentaires au texte de ce livre, notes, commentaires ou références, sont disponibles sur le site WWW de l'auteur, sous la rubrique intitulée *Systèmes d'information* :

`http://www.laurentbloch.net/`

Chapitre 1 Travail informatique : le contexte

Sommaire

1.1 À propos de conception

Les chapitres qui suivent traiteront de certaines activités des hommes, et notamment du travail, plus précisément d'un travail de conception qui par définition implique la pensée, ce qui nous conduit à consacrer quelques développements préalables à ce sujet, ainsi qu'au contexte économique contemporain où s'insère le travail des informaticiens.

Ce chapitre comportera aussi quelques remarques destinées à éclairer la psychologie de l'informaticien. Tous ceux qui ont eu l'occasion d'encadrer une équipe d'informaticiens le savent, il y a vraiment là un problème, et ils ne seront pas surpris de voir citer ici des psychiatres et des psychanalystes. L'informaticien a souvent choisi cette voie intellectuelle et professionnelle du fait d'une certaine propension à fuir la société des hommes pour celle des ordinateurs, et par là à échapper aux contraintes douloureuses de l'entrée dans l'âge adulte. Il ne s'agit que d'une tendance, plus ou moins accusée selon les individus, mais le manager qui se lancerait dans un projet informatique en l'ignorant s'exposerait à de graves désillusions.

Par cette entrée en matière, il s'agit de mieux comprendre ce qui se trame lorsqu'une entreprise demande à une équipe d'informaticiens (de la maison ou d'une société de services) de construire un pan de son Système d'Information. Si tout cela se passait habituellement sans encombre, nous pourrions nous dire que ce détour est inutile, mais, comme le savent tous les professionnels de la chose, cela se passe souvent très mal, et il convient donc de se poser quelques questions.

En arrière-plan de notre propos sur le Système d'Information (S.I.) et la démarche qui permet (ou pas) de le réaliser, se jouent notamment les questions du lien social, de la norme sociale, du pouvoir et de l'autorité, du langage et de la connaissance, enfin du travail et de la création. Il ne saurait bien sûr être question de les traiter ici en détail, mais il convient au moins de savoir qu'elles se posent, puisque aussi bien les aberrations et les échecs que nous décrirons résultent généralement de leur ignorance quand ce n'est pas de leur dénégation.

En fait, cette brève évocation de thèmes de recherche a aussi pour but de rappeler qu'un certain nombre de pratiques et de formes sociales, présentes de façon générale dans les sociétés occidentales contemporaines, et dont nous avons de ce fait souvent l'illusion qu'elles sont

inhérentes à toute société humaine, sont en réalité historiquement datées, culturellement situées, bref contingentes et non universelles. C'est important, car les collaborations fondées sur des sous-entendus sont lourdes d'échecs dès lors que ces sous-entendus se révèlent n'être pas les mêmes pour toutes les parties. Bref, une approche multi-culturelle est nécessaire pour construire le Système d'Information, parce que même si tous les participants sont citoyens d'un même pays, ils ont des origines sociales, des formations et des objectifs professionnels variés. Cela est bien sûr encore plus vrai pour un projet international.

1.2 L'économie : travail ou prestige

1.2.1 Éthique du travail dans les sociétés modernes

Les temps que l'on appelle modernes, c'est-à-dire postérieurs à la Renaissance européenne, ont vu l'autonomie de l'individu acquérir progressivement une légitimité devenue aujourd'hui totale, en contraste avec ce qui valait dans les sociétés traditionnelles, où la règle était au contraire la soumission à la tradition et à des lois transcendantes ; pour parler comme Cornelius Castoriadis [26], les sociétés traditionnelles peuvent être appelées *hétéronomes*, ce qui exprime que leurs lois et leurs règles sociales émanent d'une instance extérieure et transcendante, Dieux ou Ancêtres. Bien sûr, le fait qu'une société reconnaisse le droit à l'autonomie des individus n'empêche pas qu'une grande partie de ses membres obéissent sans grand sens critique aux règles du conformisme ambiant, alors que, dans les sociétés hétéronomes du passé comme du présent, des individus autonomes sont apparus et apparaissent, fût-ce au péril de leur vie.

Parallèlement à l'ascension de l'individu autonome, a lieu celle du travail. Dans la plupart des sociétés humaines du passé et du présent, à l'exception des sociétés développées contemporaines, le travail est considéré comme une activité répugnante et méprisable, réservée aux castes inférieures, aux femmes ou aux esclaves selon les sensibilités locales.

Les chapitres suivants du présent ouvrage seront en grande partie consacrés au travail, et plus précisément à certaines formes de travail dans le domaine de l'informatique et des systèmes d'information : il nous a de ce fait semblé difficile de ne pas consacrer ici quelques lignes à Max Weber, dont les travaux ont jeté une lumière nouvelle sur la place du travail dans les sociétés des temps modernes et de notre époque. Pour les sociétés industrialisées du XX^{e} siècle le rôle du travail semble aller de soi, ce qui introduit des illusions dangereuses dès lors que l'on s'interroge sur le travail au sein de sociétés organisées différemment, ou que l'on est amené à observer la crise des valeurs liées au travail à l'époque contemporaine. C'est là que les travaux de Max Weber nous permettent de comprendre le rôle fondamental, mais contingent, que le travail a joué dans notre société, et qu'il est en train, peut-être, de perdre.

Dans son ouvrage magistral [129], Max Weber a proposé une thèse pour expliquer comment la bourgeoisie protestante naissante du XVI^{e} siècle avait eu cette idée au départ saugrenue et qui s'est avérée géniale de faire du travail une valeur morale, économique et financière. Weber énumère les principales caractéristiques (selon lui) du capitalisme : travail et organisation rationnelle du travail, recherche du profit pour le réinvestir et non pour le dépenser. Il cite ensuite les principales caractéristiques qu'il prête au calvinisme, pour lui la forme la plus aboutie du protestantisme : ascétisme (mode de vie austère en opposition avec la religion catholique critiquée au XVI^{e} siècle par le théologien français Calvin), refus du luxe, goût de l'épargne, conscience professionnelle et croyance en la prédestination (le croyant ne sait pas s'il sera sauvé ou damné). Weber pense que ces deux systèmes de valeurs étaient particulièrement congruents et faits pour converger. Pour le capitaliste protestant, le travail, la richesse et le profit sont non seulement compatibles avec la foi chrétienne, mais peuvent devenir une force éthique contraignante. Cela suggère que la recherche du profit serait désormais devenue morale, idée que certains jugeront contestable. Cette vision du

monde a même engendré une attitude qui sous d'autres cieux ou en d'autres temps serait considérée comme une perversion : le goût de l'effort !

Malgré les disparités régionales, culturelles et individuelles que l'on peut observer, dans les pays qui ont été affectés par la Renaissance européenne et la Réforme, l'habitude s'est prise de considérer que ce qui fait la valeur d'une personne, c'est ce qu'elle a accompli, réalisé, qui le plus souvent est de l'ordre du travail, étant entendu que la création artistique ou l'entraînement sportif sont du travail. Pour l'ancienne échelle de valeurs aristocratique, le mérite personnel n'est rien, la qualité d'une personne est déterminée par sa naissance, éventuellement embellie par d'autres qualités (aussi peu laborieuses que possible), dont l'œuvre de Marcel Proust dresse un catalogue raisonné et critique, mais cette vision du monde nous apparaît aujourd'hui comme une survivance pittoresque digne du musée.

Or dans tout ce que l'on appelait hier le Tiers-monde l'ancienne vision du travail subsiste. Psychanalystes plongés à la fin des années 1960 dans une pratique thérapeutique hospitalière au Sénégal, Marie-Cécile et Edmond Ortigues [88] écrivent : « Le vœu d'affirmation virile exprimé, exprimable, n'a ni la même forme ni le même contenu que ceux que nous lui connaissons en Europe. En Europe, le vœu du jeune Œdipe est de rivaliser dans des tâches, des actions, des réalisations. La rivalité se cherche une sanction objective, elle se pense médiatisée. Ici [à Dakar, *Ndla*] l'accent est davantage mis sur l'affirmation d'un statut, d'un prestige…

L'activité précise, disons le métier, que suppose la bonne situation ou l'acquisition de beaux vêtements, est peu considérée pour elle-même. Le vœu est moins celui d'une activité plus intéressante ou plus efficace que d'une place plus en vue, d'une raison sociale plus éminente. » (pp. 122-123).

Dans les pays où l'économie de marché a atteint un certain développement, disons pour résumer les pays de l'OCDE, l'activité humaine est largement orientée par des principes qui correspondent aux descriptions et aux analyses de Max Weber, notamment l'organisation rationnelle du travail et la recherche morale du profit. Sous d'autres climats nous pouvons observer la prééminence d'autres modèles d'organisation sociale, qui étaient d'ailleurs les nôtres il n'y a pas si longtemps, et dont les survivances sont encore nombreuses dans un pays comme la France. Beaucoup de pays sont en effet régis par des systèmes politiques autocratiques qui ignorent l'État de droit, pour ne rien dire de la séparation des pouvoirs. Guy Sorman a donné une description brève mais réaliste d'un tel régime [103]. Le succès d'une entreprise économique dans un tel système dépend beaucoup moins de la qualité de sa direction, de son organisation et du travail de ses employés que de la faveur du pouvoir politique. L'autocrate ne tolère aucune activité indépendante de son bon vouloir, et il fait sentir son pouvoir par l'arbitraire qu'il exerce, dont il résulte que la fortune d'un jour peut être anéantie le lendemain. Cette conception du pouvoir se propage à tous les échelons de la société, et chaque responsable de niveau intermédiaire tient à exhiber son rang par des signes extérieurs de munificence (voitures, richesse du décor, domesticité…) et par la morgue qu'il manifeste envers ses subordonnés et qu'il adoucit périodiquement par des marques de paternalisme. Dans un tel contexte, travailler d'arrache-pied serait bien sûr une erreur complète, d'ailleurs rarement commise : il est bien plus judicieux de consacrer son énergie à des pratiques courtisanes pour devenir membre de la clientèle d'un patron puissant. En fait ne travaillent que ceux qui peuvent y être obligés par la contrainte physique : ouvriers agricoles et du bâtiment, employés des nouvelles industries taylorisées. Nous aurons l'occasion de revenir sur cette propriété du travail manuel élémentaire, de pouvoir être imposé par la force, ce qui est en fait toujours le cas.

1.2.2 Le sous-développement dans les sociétés développées

Le lecteur informé n'aura pas manqué de reconnaître, dans la description du sous-développement économique, politique et social ébauchée à l'alinéa précédent, tel ou tel

trait de certains secteurs de notre propre société, spécialement dans les administrations et les entreprises publiques. Ce n'est d'ailleurs pas sans inquiétude que l'on observe le développement du caractère arbitraire et subjectif associé à ce type de management dans les grandes entreprises privées où le pouvoir a été pris par des financiers à courte vue, l'œil braqué sur le résultat trimestriel. Jusqu'à une époque récente, la prévisibilité, pour un salarié, consistait à se dire que s'il faisait son travail correctement, c'est-à-dire s'il contribuait plus aux profits de l'entreprise qu'à ses pertes, sous réserve que l'activité globale soit profitable, sa situation était sûre ; ce sentiment de sécurité était certes souvent trompé, mais globalement, dans notre pays en tout cas, dans les secteurs économiques non sinistrés, c'était ainsi ; on pourrait appeler cela le contrat social du capitalisme, instauré à l'issue de décennies de luttes et de négociations. Aujourd'hui il a volé en éclats. Les financiers exigent des taux de profit de l'ordre de 15 à 20% dont il est clair qu'ils sont totalement irréalistes sur plusieurs années consécutives. Les managers s'efforcent d'atteindre cet objectif par ce qui s'apparente à de la « cavalerie » : achat d'entreprises, ventes d'actifs, délocalisations, fermetures d'usines ou de filiales, investissements dans des secteurs encouragés par des subventions ou des avantages fiscaux, opérations monétaires ou boursières dont la nature particulière retient parfois l'attention de la justice, mouvements tactiques de trésorerie, tout cela se traduisant par des cessations d'activité et des licenciements souvent assez arbitraires. Dans un tel contexte, il apparaît clairement que le meilleur moyen de sauver sa peau n'est en aucun cas le travail consciencieux, mais bien plutôt l'entretien d'un réseau de relations bien choisies, l'intrigue, le complot et, surtout, une présence courtisane auprès du dirigeant actuel ou futur. Cela s'appelle le management guidé par les affects.

Un aspect particulièrement pernicieux de cette perte de repères a été décrit par le sociologue Jean-Pierre Le Goff [70]. Je reprends les citations qu'il fait (pp. 19-20) d'un manuel produit par EDF-GDF [47] : ses auteurs entendent établir une « relation de confiance réciproque » qui « provient davantage d'un lien émotionnel fort que d'un raisonnement intellectuel et d'une adhésion rationnelle » (p.63). Dans cette perspective, les nouveaux managers se doivent de « changer l'atmosphère de travail », « modeler le comportement des individus » (p. 123) ; véritables « ingénieurs des âmes » de l'entreprise, leur mission est de susciter chez les employés « un sentiment d'émotion et de satisfaction qui les lie à l'entreprise par une relation nettement plus saine et plus sincère que la dépendance dans laquelle les plonge la sécurité de l'emploi » (P. 253). Ce qui gît sous ce projet suave de conférer aux relations de travail un érotisme polymorphe, c'est le règne de la subjectivité et de l'arbitraire et l'abolition de toutes les règles de comportement auxquelles pouvaient s'obliger réciproquement patrons et employés, règles élaborées, nous l'avons déjà dit, à l'issue de décennies de conflits et de négociations.

Il est possible d'observer cette perversion dans d'autres domaines, et spécialement dans le champ de la « formation » destinée selon certains à succéder à l'éducation et à l'instruction. J'ai ainsi relevé en janvier 2004 sur le site de l'*Association francophone pour la recherche et l'enseignement en sciences et technologies de l'information* un article qui cite [18] quelques déclarations croustillantes d'experts en télé-formation :

« La neuvième édition de *E-learn Expo*, quatrième exposition à Paris, est le reflet des turbulences et des consolidations du marché, comme le montre la fusion de *Docent* avec *Click2learn*.

Andy Sadler (IBM États-Unis) a présenté l'apprentissage juste-à-temps, sur le lieu de travail *"On Demand"*. L'école est finie, a-t-il dit, et il est besoin d'un apprentissage personnalisé avec soutien sur le lieu de travail, l'environnement d'apprentissage étant complètement intégré au flux de travail, "à la demande".

Selon une récente étude menée par Thomson NETg et le cabinet international de conseil en gestion des ressources humaines, DBM, un employé sur cinq seulement est parfaitement qualifié pour le poste qu'il occupe. Ces résultats mettent en évidence le fait qu'aujourd'hui les entreprises négligent les besoins en formation de leurs collaborateurs. En

conséquence, elles réduisent, en même temps que leur propre productivité, la motivation et la satisfaction de leurs employés.

Près de 40% des personnes interrogées ont déclaré qu'une formation et une évolution plus motivantes contribueraient à améliorer la satisfaction qu'elles retirent de leur travail, tandis que 24% pensent qu'un *coaching* régulier en améliorerait la qualité. Plus de la moitié (60%) des personnes interrogées, en outre, ne perçoit pas clairement l'adéquation entre la formation dispensée et les objectifs stratégiques globaux de leur entreprise. »

Quiconque possède la moindre expérience éducative mesurera la vacuité de ces propos pour ce qui est de l'augmentation des compétences et des savoirs de ceux qui sont censés en subir les effets, en revanche leur application à ce que l'on désigne par l'ignoble locution « gestion des ressources humaines » (a-t-on seulement pensé au sens de ces mots ?) risque d'advenir avec des effets saisissants !

1.2.3 Crise du travail

Les lignes qui précèdent font plus que suggérer que le travail, qui était une innovation révolutionnaire au XVI^e^ siècle et le principe de l'organisation sociale au XX^e^ (il en sera plus amplement question ci-dessous au chapitre 2), est entré dans une phase de crise et de déclin. Certains n'hésitent pas à prédire sa disparition. C'est d'ailleurs la raison qui a poussé l'auteur à écrire ce livre, et qui justifie qu'avant d'entrer dans le vif du sujet il lui faille passer par ces préliminaires : si savoir de quoi on parle lorsque l'on dit « travail de réalisation d'un système informatique » allait de soi, ce livre n'aurait pas de raison d'être. C'est parce que j'ai observé pendant trente-cinq ans les erreurs lourdes et coûteuses où menait la confusion dans ce domaine que j'ai entrepris de l'écrire. Ces ravages ont souvent été provoqués par des modes managériales perverses ou intéressées, et ces modes ont souvent pu se répandre grâce à la désinformation du milieu auquel elles étaient destinées : c'est pourquoi j'ai cru utile de passer en revue un certain nombre de sujets dont il faut au moins connaître l'existence pour pouvoir réfléchir sérieusement au travail, et qui sont proposés ici au lecteur comme autant de pistes à explorer.

1.2.4 Institutions, travail, argent

Une série d'évolutions sociales, idéologiques et intellectuelles contribuent toutes concurremment à une certaine perte de repères qui affecte la vie en société dans les pays occidentaux. Cette dissolution du lien social n'est pas anecdotique, elle ne se réduit pas à quelques problèmes d'autorité au sein de la famille ou d'éducation trop laxiste, et Jean-Claude Guillebaud [55] a raison de souligner qu'il ne saurait y avoir de réponse aux questions qu'elle nous pose sans la prise en considération de l'histoire apocalyptique du XX^e^ siècle, commencé avec les génocides des Herreros de Namibie et des Arméniens et la boucherie de 1914-1918 pour se terminer avec le génocide des Tutsis du Rwanda, qui encadrent les tyrannies et les massacres les plus épouvantables de l'histoire de l'humanité, sans oublier les massacres coloniaux qui en avaient constitué la répétition. En effet il y a dans cette histoire de quoi dissoudre pas mal de choses, notamment l'optimisme et la confiance en l'individu. Le présent ouvrage ne se propose pas de répondre à ces questions qui le dépassent, mais la situation d'affaiblissement du lien social nous intéressera ici surtout pour ses répercussions dans le monde du travail.

Cornelius Castoriadis, dans son livre *La montée de l'insignifiance* [26], signale que « le capitalisme n'a pu fonctionner que parce qu'il a hérité d'une série de types anthropologiques qu'il n'a pas créés et n'aurait pas pu créer lui-même : des juges incorruptibles, des fonctionnaires intègres et wébériens, des éducateurs qui se consacrent à leur vocation, des ouvriers qui ont un minimum de conscience professionnelle, etc. Ces types [...] ont été créés dans des périodes historiques antérieures, par référence à des valeurs alors consacrées et incontestables. » (p. 68). L'inaptitude croissante à créer ces types humains, leur destruction

même, va de pair avec l'essor de l'individualisme et l'épuisement de notre stock de convictions partagées, dont des événements impensables ont ruiné tout le potentiel d'adhésion unanime ou tout au moins majoritaire.

L'expérience de l'auteur lui suggère que cette perte de repères est particulièrement sensible dans les univers abrités des aspects ordinaires des relations entre employeurs et employés, tels que le risque de chômage et de faillite de l'entreprise. En France spécialement, le monde de la fonction publique est totalement à l'abri de ces aléas, et certains de ses secteurs, comme l'enseignement et la recherche scientifique, conjuguent (pour les personnels dotés d'un statut de titulaire) cette protection sociale avec un niveau de vie relativement élevé par rapport à la moyenne nationale et une forte endogamie pratiquée souvent depuis plusieurs générations. La conjonction de ces facteurs contribue dans les populations concernées à une relative érosion de la conscience de la réalité des relations entre employeur et employé, et notamment des aspects économiques de cette réalité. Il ne nous a pas paru totalement futile d'y consacrer un développement.

Il ne fait guère de doute que pour beaucoup de gens qui n'ont pas la chance d'avoir un travail passionnant où leur esprit d'initiative est fortement encouragé, la fréquentation d'un club de vacances comporte bien des aspects agréables en comparaison de celle des locaux où leur employeur les accueille. Les horaires des activités sont assez souples, leur nature est variée et il est loisible d'en changer à sa guise. Toutefois, le flux financier entre l'institution et la personne affiliée n'a pas le même volume ni surtout la même orientation. Si l'on regarde d'un peu plus près, il y a aussi la question de la prise de décision : le gentil membre dispose d'une grande latitude dans l'expression de ses désirs relatifs à la nature des activités, et il peut, sans trop de difficulté, influer sur des décisions qui contribueront à la satisfaction de ses désirs. Dans une entreprise, ce sont généralement les actionnaires et les dirigeants qui se réservent la possibilité de choisir la nature des activités proposées aux salariés ainsi que les modalités selon lesquelles elles seront accomplies. Ce pouvoir a, au fil des ans, été tempéré par les lois sociales, qui prévoient notamment l'information et la consultation des salariés, mais globalement et pour dire les choses avec ce qui paraîtra à certains une certaine brutalité, c'est le patron qui décide ce que fait l'entreprise et, en échange de leur salaire, il achète aux employés leur contribution à ce qu'il a décidé. Si l'employé ne souhaite pas contribuer au plan du patron, le contrat qui les lie sera rompu, à l'initiative de l'un ou de l'autre.

Enfin, en principe le monde marche ainsi. La teneur de l'alinéa précédent allait sans doute de soi il y a une cinquantaine d'années : si le principe en reste solide, son application est plus complexe aujourd'hui. Ce que les employeurs attendent des salariés est devenu beaucoup plus difficile à définir, et surtout à quantifier. La façon dont les salariés remplissent leur mission s'est compliquée. Pour beaucoup d'entre eux la tâche qui leur incombe comporte une part importante de collaboration avec d'autres personnes (éventuellement extérieures à l'entreprise), de réflexion, d'apprentissage et de recherche d'information au sens large, toutes choses dont le contrôle est difficile. Répondre clairement à la question de ce que c'est, travailler, n'est pas aussi facile ici que dans le cas où cela se mesure en nombre d'hectares labourés ou en nombre d'heures passées sur une chaîne. Imaginons simplement ce qu'un parent moderne peut répondre à son enfant qui lui demande ce qu'il fait au travail : pour beaucoup de salariés contemporains la réponse ne va vraiment pas de soi. Pour des raisons faciles à comprendre les employeurs cherchent volontiers à ramener les cas complexes à des cas simples, mais souvent, et notamment dans les cas qui font l'objet de ce livre, cet effort de simplification mène à des erreurs (voir ci-dessous, chapitre 2). Bref, la réalité des relations de travail et les rôles que chacun y jouent sont loin d'être clairs aujourd'hui, il suffira pour s'en convaincre de jeter un coup d'oeil au petit livre de Corinne Maier qui a défrayé la chronique médiatique [78].

1.2.5 Classification des lieux d'activité

La méconnaissance éventuelle des réalités du travail par certains jeunes lecteurs encore peu initiés aux cruautés du monde me donne à penser qu'un tableau descriptif de quelques institutions prises en exemple pourrait ne pas être tout à fait inutile.

Les institutions citées dans ce tableau n'ont pas grand-chose à voir avec les institutions politiques, et notamment la démocratie, au sens de démocratie politique[1], ne semble pas un principe d'organisation pertinent en ce qui concerne du moins le cœur de leur activité. Nous pouvons imaginer que dans la maison de retraite le choix du programme de télévision au salon collectif soit effectué selon une règle démocratique, mais on voit bien que ce n'est pas là que se jouent la nature et le rôle fondamental de l'institution. Chacune des institutions évoquées ici est constituée dans un but précis et s'assure la présence et la contribution des agents nécessaires à son activité par les moyens à sa disposition : contrainte physique ou légale, persuasion, appât du gain, promesse de bénéfices futurs ou de plaisirs immédiats, influence morale...

Institution	Caractère volontaire de la présence et de l'activité.	Orientation du flux monétaire
Club de vacances Maison de Jeunes	100 %	individu → institution
Lycée	plus ou moins, activités imposées	individu → institution (modéré)
Prison	pas du tout	individu → institution (c'est selon)
Religion	c'est selon	individu → institution (modéré)
Maison de retraite	c'est selon, activité libre	individu → institution (significatif)
Entreprise	présence libre, activité encadrée	institution → individu (significatif)
Armée (1914-1918)	présence et activité imposées par le peuple souverain et encadrées	institution → individu (symbolique)
Plantation (esclavage)	présence et activité imposées et encadrées par personne privée	institution → individu (presque nul)

En revanche la liberté est ici un paramètre très important. Ainsi l'usine capitaliste moderne et la plantation esclavagiste ne sont pas sans partager quelques traits communs, mais la liberté du travail dans la première est une différence essentielle. De même, la contrainte exercée sur le soldat de la guerre de 1914-1918 pouvait ressembler à celle qui pèse sur l'esclave, si ce n'est que ce soldat appartenait lui-même à la collectivité qui avait décidé la guerre et qui de ce fait exerçait la contrainte en question, ce qui changeait tout. On ne manquera pas d'observer d'ailleurs que cette distinction était beaucoup moins nette pour les soldats des troupes coloniales, dépourvus de droits civiques et enrôlés de force.

1 Il convient de distinguer la *démocratie sociale*, qui consiste en ce qu'aucune fonction sociale n'est réservée à une catégorie particulière de personnes distinguées par leur naissance, de la *démocratie politique*, qui établit le pouvoir du peuple, c'est-à-dire le fait que le peuple crée les lois.

1.2.6 Retour à Malthus ?

En fait les phénomènes de pression sur la force de travail que nous constatons s'inscrivent dans un mouvement plus général qui a contaminé l'économie mondiale au cours des trente dernières années, c'est-à-dire depuis le premier choc pétrolier de 1974. Sans prétendre en faire ici une analyse complète, nous en relèverons quelques traits.

Dans les entreprises, les mots d'ordre sont rationalisation et réduction des coûts, cela dans un contexte économique étendu à la planète.

Essayons de voir ce qu'il en est. Après tout, l'organisation économique qui prévalait au lendemain de la seconde guerre mondiale n'était pas idyllique et n'instaurait pas le paradis sur terre, et le développement économique récent a produit des augmentations massives du niveau de vie dans de nombreuses régions du globe. Ces effets du développement sont inégalement répartis : l'Afrique a vu sa situation empirer, et certains pays s'y enfoncent dans la catastrophe. L'Asie méridionale et orientale offre un tout autre tableau, plusieurs pays y connaissent une croissance industrielle considérable avec un paysage social qui n'est pas sans rappeler l'Europe occidentale du XIX^{e} siècle : capitalisme sauvage, spéculation débridée, crises dramatiques, fortes inégalités sociales, conditions de travail peu acceptables, mais au bout du chemin la sortie de la misère rurale qui était il y a cinquante ans comparable à celle de l'Afrique, avec des famines et des épidémies catastrophiques. Il suffit de lire Balzac et Stendhal pour savoir que la France est passée par là.

Modes du management

Il est instructif d'examiner quelles sont les diverses manières d'appliquer les mots d'ordre de rationalisation et de réduction des coûts, et quelles en sont les conséquences à court et moyen terme, à petite et grande échelle.

Il y a eu la vague « zéro stock, production juste à temps », qu'un journaliste facétieux a traduit par « rien ne sert de tuer l'ours avant d'avoir vendu sa peau », ce qui semble intelligent, mais reste encore à tuer l'ours, ce qui n'est pas forcément le plus facile. À l'échelle d'une entreprise et à court terme, cette doctrine semble très avantageuse. Les stocks sont coûteux, et la versatilité de la clientèle fait toujours craindre les invendus. Il n'est d'ailleurs pas faux que la constitution de stocks excessifs soit de mauvaise gestion, et que la personnalisation des commandes des clients soit un apport utile de l'informatisation, mais, comme pour beaucoup d'autres méthodes que nous examinerons, une bonne idée appliquée avec pragmatisme peut devenir source de catastrophes dès lors qu'elle est érigée en dogme.

Il est aussi devenu à la mode de sous-traiter la plus grande partie des opérations de production, ce qui reporte sur les sous-traitants les tensions imposées par le « juste à temps » et par les stocks. Et le tout sera couronné par le recours massif au personnel intérimaire. On peut aussi délocaliser, ce qui présente les avantages conjugués de la sous-traitance et de l'intérim, avec en outre des coûts de main d'œuvre abaissés, mais sans doute avec un peu plus de stocks...

Si l'on regarde les perspectives offertes par une telle politique à un horizon d'un an et pour la comptabilité d'une entreprise, tout cela semble cynique mais judicieux. Nul doute que le manager qui aura appliqué toutes ces recettes, apprises l'année précédente dans son école de gestion, aura de la promotion et des stock-options.

Maintenant essayons de voir ce qui se passe à l'échelle macro-économique, si toutes les entreprises se rallient à cette doctrine promue au rang de dogme absolu (et c'est bien ce qui s'est passé). Le « juste à temps » engendre des coûts de transport élevés, et, si l'on prend un peu de hauteur pour être plus « macro », des coûts de construction et d'entretien d'infrastructures routières, ferroviaires, portuaires et aéroportuaires qui seront bien un jour ou l'autre répercutés à l'ensemble des acteurs économiques, donc aux entreprises, sans parler des effets sur l'environnement. De plus, « juste à temps » se traduit souvent (très souvent)

par « livré en retard », avec les effets en cascade qui en découlent pour tous les agents économiques en aval : projets différés, commandes annulées, pénalités de retard, personnel et sous-traitants rémunérés l'arme au pied... De façon similaire, l'emploi intérimaire ou en contrat à durée déterminée a un coût horaire beaucoup plus élevé que l'emploi en contrat de travail à durée indéterminée (selon la législation française), surcoût compensé par une plus grande souplesse dans l'adaptation des effectifs aux besoins. Cet avantage est donc surtout sensible pour de courtes périodes de travail, beaucoup moins si ces formes d'emploi sont utilisées à longueur d'année, d'autant plus qu'apparaissent alors des inconvénients insidieux : perte de compétence de l'entreprise, état d'esprit « mercenaire »... On peut penser que la perte de confiance de l'employé ainsi traité représente aussi un coût pour l'employeur, mais il n'est pas facile à calculer.

Dès lors que toutes les entreprises agissent de la même manière, les avantages attendus sont compensés au moins en partie par des inconvénients qui frappent tout le monde, simplement les avantages sont plus faciles à observer que les inconvénients. L'entreprise a bien réussi à externaliser des coûts, mais si tout le monde en fait autant chacun paye pour les externalisations de tout le monde. On peut aussi étrangler les sous-traitants ou les transporteurs : le résultat est connu, il s'appelle naufrage de l'Erika, pas franchement bon marché ni très glorieux. Bref, il y a des effets d'aubaine ponctuels mais pas de vraie réduction de coûts à l'échelle globale, compte tenu notamment des sommes énormes consacrées aux réorganisations permanentes.

Il y a aussi le fantasme de l'entreprise industrielle « sans usine », très populaire en France, pays qui n'a jamais aimé son industrie[2]. La seule ombre au tableau, c'est que dans les industries de pointe (micro-électronique, aérospatial, télécommunications) la valeur ajoutée et le savoir-faire sont beaucoup dans les procédés de fabrication. On sait trop peu qu'une usine de production de microprocesseurs analogues à ceux qui animent un banal ordinateur de supermarché coûte, selon la taille, de quatre à quinze milliards d'Euros (il s'agit bien de milliards, pas d'une faute de frappe). Une entreprise qui ne construit pas de telles unités de production (et qui donc *ne sait pas* en construire) ne saurait jouer un rôle de premier plan dans l'industrie électronique.

L'exemple de *Dell Computer*

L'entreprise *Dell Computer* fut créée au Texas en 1985 par Michael Dell. En 2003 elle fut sans doute le principal vendeur d'ordinateurs personnels du monde avec 43 milliards de dollars de chiffre d'affaires. Voici des chiffres plus récents à titre de comparaison :

	CA 2016 (milliards $)	Effectifs	Immobilisations (milliards $)	R&D 2013 (milliards $)
Intel	59	106 000	113	10,1
Dell	62	138 000	118	0,9
IBM	80	380 000	117	6,3
HPe+HP	98	429 000	146	3,6
Oracle	37	136 300	112	4,5
Microsoft	85	114 000	194	9,8
Google	90	57 000	167	8,1
Facebook	28	18 800	65	2,2
Apple	216	116 000	322	3,8

On trouvera une analyse intéressante de son activité dans un rapport de recherche de l'Université de Californie à Irvine [65], auquel nous empruntons une grande partie des informations relevées ici, les autres m'ayant été fournies par mon collègue Patrick Lerouge

2 Que l'on compare simplement, pour s'en convaincre, les poids politiques respectifs du Ministère de l'Agriculture et du vague sous-secrétariat d'État au rattachement variable qui s'occupe de l'industrie.

de l'Inserm qui a été invité à un voyage d'étude en Irlande sur les sites de Dell et d'Intel, ainsi que par le site WWW de Dell.

À l'inverse des autres entreprises évoquées ci-dessus, Dell ne possède en propre aucune technologie et ne fabrique rien, mais assemble des éléments achetés à d'autres entreprises. Ces éléments peuvent être individuels, par exemple des moniteurs, des disques ou des châssis, mais aussi des sous-ensembles déjà composites, comme une carte-mère équipée d'un processeur, de mémoire, d'un processeur graphique et d'une interface réseau. Certains ordinateurs portables sont même carrément achetés tout faits à des entreprises comme Samsung, Quanta et Compal.

Les usines Dell sont des centres d'assemblage, où un ordinateur banal est assemblé en trois minutes, y compris les premiers tests, par exemple dans les deux usines de Limerick en Irlande. Les autres centres de production sont près d'Austin au Texas, à Nashville dans le Tennessee, Eldorado do Sul au Brésil, Penang en Malaisie et Xiamen en Chine. Les fournisseurs et les sous-traitants (à l'exception d'Intel, en position de force monopolistique pour les processeurs) sont tenus d'avoir des stocks dans des camions stationnés en permanence sur le site Dell, et du personnel pour décharger les camions au fur et à mesure de l'avancement de la production. Le matériel livré ne devient la propriété de Dell qu'après le déchargement, sauf pour les processeurs Intel qui sont la propriété du client dès la sortie de l'unité de production Intel.

Un autre point fort de Dell est son modèle de vente, qui repose à 90% sur la relation directe avec le client final, plutôt que sur un réseau de distributeurs, et aujourd'hui beaucoup sur la vente en ligne. Dans un pays comme la France notamment, Dell a su capter la clientèle de beaucoup d'administrations publiques et d'organismes de recherche.

Dell est incontestablement le plus beau succès des méthodes de management exposées ci-dessus appliquées de façon systématique et extrême. Traits caractéristiques, ses coûts logistiques sont nettement supérieurs à ses coûts de fabrication, et ses critères pour ses implantations mettent au premier plan la qualité des infrastructures de transport, à côté d'autres critères comme le régime fiscal, les aides locales à l'installation, l'appartenance à des zones de libre échange (Union Européenne, Mercosur) ou d'unité monétaire (zone Euro), et la faiblesse des syndicats. Dell ne recherche pas particulièrement la proximité de « *clusters* » industriels tels que la *Silicon Valley*, de centres de recherche fondamentale ou de centres de formation d'ingénieurs et d'universitaires, où le personnel hautement qualifié abonde mais où les coûts de main-d'œuvre sont plus élevés.

Le revers de cette réussite est son extrême dépendance à l'égard des créateurs de la technologie vendue avec un tel succès par Dell. La faiblesse des investissements productifs de Dell est à comparer au prix colossal des unités de production d'Intel, par exemple, dont l'usine irlandaise a coûté 15 milliards de dollars. Si l'on regarde les budgets de recherche-développement, le chiffre pour Dell est de l'ordre de 500 millions de dollars par an, soit 1,2% du chiffre d'affaires, et pour Intel le chiffre est de 4,4 milliards de dollars, soit 14% du chiffre d'affaires. Cette différence est peut-être à l'avantage de Dell du point de vue du rentier qui achète des actions, mais elle désigne bien l'endroit où se crée la plus-value la plus importante. Intel prend sûrement plus de risques, mais a sans doute des bases plus solides : c'est une entreprise qui oriente les évolutions de la technologie, alors que Dell s'y adapte, avec bonheur jusqu'à présent, mais cela pourrait changer.

Dell peut craindre de connaître le sort des fabricants d'ordinateurs compatibles avec les grands systèmes IBM, nés comme des champignons dans les années 1970, extraordinairement prospères durant une vingtaine d'années, puis disparus sans laisser de traces après qu'IBM eut réorganisé sa technologie et sa propriété industrielle pour les torpiller sans prêter le flanc aux législations anti-trust.

Modes du management, suite

En poursuivant notre visite du sottisier managérial[3] accumulé au cours des décennies récentes, nous trouvons l'« organisation de l'entreprise en centres de profit ». Je l'illustrerai d'un exemple : dans les années 1970 l'acheteur d'un système informatique (c'était gros et cher) se voyait proposer, à l'issue de la période de garantie, un contrat de maintenance. Le service de maintenance du fournisseur avait reçu du service des ventes de systèmes la description détaillée de la configuration livrée et pouvait donc faire une proposition commerciale adaptée. Au début des années 1980, les fournisseurs se sont réorganisés en « centres de profit », c'est-à-dire que le service de maintenance et le service des ventes de systèmes ont commencé à se comporter comme des entreprises indépendantes. Le résultat le plus clair pour le client, d'expérience vécue, fut que s'il voulait un contrat de maintenance c'était à lui de réaliser l'inventaire de la configuration du système qu'il venait d'acquérir en respectant bien les nomenclatures multiples et changeantes du constructeur, une tâche non triviale, pour passer commande au service de maintenance. Les erreurs ou omissions l'exposaient à avoir des matériels sans maintenance, qui en cas de panne coûteraient fort cher à réparer.

À courte vue, ce principe d'organisation est génial : on reporte sur le client le travail d'organisation et de coordination des divers services de l'entreprise. Avec un peu de recul c'est complètement contre-productif, et cela n'a pas peu contribué au déclin des services de maintenance.

En fait, la cible de telles réorganisations est le plus souvent interne à l'entreprise : il s'agit d'exacerber la concurrence entre managers et de lutter contre le confort qui risquerait de s'installer en l'absence de ces réorganisations permanentes qui remettent sans arrêt en question les positions des uns et des autres. Le défaut de ces pratiques, c'est que, malgré les proclamations contraires, le souci du client en est complètement absent, c'est un pur accroissement de l'entropie locale.

Parmi ces sottises j'ai une tendresse particulière pour le « recentrage sur le métier de base », rebaptisé ci-dessus « persévérer dans la routine et mettre tous ses œufs dans le même panier ». Michel Volle (dont le site WWW [119] est une mine de données et d'analyses sur tout ce qui a trait aux systèmes d'information) me faisait d'ailleurs remarquer que cette notion de « métier de base » était très floue : quel est le métier de base d'Air France ? faire voler des avions ? vendre des voyages ? transporter des gens et des choses ? Quoi qu'il en soit, une entreprise qui vise le développement à long terme doit au contraire se diversifier techniquement, géographiquement et commercialement, innover et créer. D'ailleurs les entreprises dirigées non par des bureaucrates irresponsables issus de la finance ou, en France, de la fonction publique, mais par leur propriétaire, telles *General Electric* ou en France le groupe Bouygues, se gardent bien d'obéir aux canons de la gestion à la mode. Le problème, c'est qu'une fois qu'une entreprise est tombée entre les mains des institutions financières elle n'a plus guère le choix, elle doit se plier à un modèle d'organisation accessible aux capacités intellectuelles des financiers, donc se cantonner à la mono-activité. J'ai lu ce diagnostic dans l'interview par Irène Inchauspé [59] d'un homme peu suspect d'être un trublion altermondialiste, Michel David-Weill, dirigeant depuis trente ans de la banque Lazard Frères.

Posons-nous la question de l'existence d'une idée unificatrice entre ces quelques pratiques que nous avons brièvement décrites, et dont le lecteur pourra facilement accroître la collection en pénétrant dans une librairie d'aéroport et en feuilletant quelques volumes présentés sur la table consacrée aux nombreux livres dotés de titres du genre *Comment devenir*

3 J'y insiste encore : les méthodes décrites ici et d'ailleurs dans l'ensemble de ce livre comme des aberrations reposent toutes, au départ, sur du bon sens et de bonnes idées. C'est leur application systématique qui en fait des catastrophes. On trouvera une synthèse sur la question des modes managériales chez Michel Villette[115].

un manager à succès sans se fatiguer et sans rien savoir, ou en s'inscrivant aux cycles d'enseignement prétendu supérieur de plus en plus nombreux qui correspondent à ce programme séduisant. Il y a une idée commune : elle est vieille et usée, c'est ce brave malthusianisme, toujours ridiculisé et chaque fois ressorti de la poubelle où on l'avait jeté.

Le malthusianisme correspond à la vision d'un monde en rétrécissement, où les ressources disponibles diminuent et où il faut faire des économies, restreindre les dépenses, « mettre de côté ». C'est le contraire d'une vision d'entrepreneur, qui investit pour développer à terme des capacités de production qui élargiront le monde. C'est une pensée de rentier. On appelle aujourd'hui investisseurs les gens qui achètent des actions : c'est un contresens. Il faudrait les nommer rentiers. Les investisseurs sont ceux qui créent les capacités de production. Hervé Le Bras a dressé un tableau chatoyant de la pensée malthusienne, trop souvent considérée comme purement démographique alors qu'elle est avant tout économique [68].

Le paradoxe du monde économique contemporain, c'est d'être mené par des financiers malthusiens qui exigent des entreprises par eux possédées des taux de profit de l'ordre de 15 ou 20%, comme déjà noté plus haut. Comme aucune entreprise ne peut tenir ce rythme à long ou même à moyen terme, lorsque les diverses occasions de cavalerie financière, de ventes d'actifs et d'autres profits faciles qu'elle offrait sont épuisées, elle est démantelée et jetée. Bref, sous couvert de réduction des coûts c'est un gigantesque gaspillage qui s'installe ; lorsque des entreprises aussi solides que Panam ou Swissair sont mises à l'encan, c'est une perte irréparable qui n'est compensée par aucun gain, hormis l'intérêt de quelques opérateurs financiers.

1.2.7 Travail et imitation de travail

Cet examen des pratiques néo-malthusiennes qui, sous prétexte d'efficacité et de rentabilité, organisent le gaspillage et le désinvestissement, nous mène à Alexandre Zinoviev, professeur de philosophie et de logique à Leningrad (aujourd'hui Saint-Pétersbourg), expulsé d'URSS en 1978 après la parution clandestine de son livre *Les hauteurs béantes* [134], et revenu en Russie en 1999. Dans cet ouvrage, qui mêle la fiction à la philosophie et à l'analyse politique, Zinoviev élabore une théorie du fonctionnement de la société soviétique dont beaucoup d'aspects s'appliquent bien à tout univers bureaucratique, qu'il s'agisse des états-majors des grandes entreprises multinationales, des universités ou des administrations centrales du secteur public français.

Une des thèses les plus pénétrantes de Zinoviev consiste à distinguer le travail de l'imitation de travail (p. 216-218).

« Le travail nécessite souvent peu de monde (parfois deux ou trois, ou à la rigueur cinq personnes). L'imitation du travail mobilise de grandes masses de gens, qui peuvent se compter par dizaines et par centaines...

Bien souvent, le travail peut être fait en quelques jours ou en quelques mois. L'imitation du travail peut durer des années ou des décennies entières...

Le travail est discret, banal, ennuyeux. Il est laborieux. L'imitation est faite d'agitation. On peut la figurer comme une immense représentation théâtrale. Ce sont des réunions, des symposiums, des rapports d'activité, des voyages, des luttes groupusculaires, des remplacements de direction, des commissions, etc. »

Zinoviev applique ce concept d'imitation de travail à l'analyse de la société soviétique dans son ensemble et aussi, de façon plus particulière, au monde de la création intellectuelle, de la recherche scientifique et des universités. On pourra penser à un rapprochement avec le chef-d'œuvre de Robert Musil, *L'homme sans qualité*, et à sa description ironique, acerbe mais pleine d'humanité de l'*Action parallèle*, regroupement de dames aristocratiques et d'intellectuels mondains pour améliorer le climat spirituel de l'empire austro-hongrois, alors que l'on est à la veille de la guerre de 1914-1918 et à l'avant-veille de l'ascension du nazisme.

Le lecteur qui aurait fréquenté certains états-majors du secteur tertiaire, certaines administrations centrales ou certains milieux académiques conviendra sans doute que la description de Zinoviev ne s'applique pas uniquement au système bureaucratique soviétique. Lorsque, après avoir fréquenté pendant plusieurs années le monde des administrations financières françaises, j'ai lu ces lignes – je ne saurais trop conseiller au lecteur de se reporter au texte intégral – j'ai eu l'impression précise de lire une analyse pénétrante d'une situation vécue des dizaines de fois. Depuis lors, d'innombrables réunions de groupes de gestion de projet ou de commissions des marchés ont enrichi ma collection d'imitations de travail, et la suite de ce livre ne manquera pas d'y puiser.

Chapitre 2 Définition et contrôle du travail à faire

Sommaire

Dès lors que le travail est une marchandise qui s'achète et qui se vend se posent à son sujet les questions économiques classiques de la valeur, du prix et de la quantité, ainsi que de leur contrôle. Pour le travail informatique qui nous intéresse ici, ces questions n'ont jamais trouvé de réponse pleinement satisfaisante. Classiquement, la quantité de travail se mesure par le temps de travail : c'est cette quantification qui se révèle de plus en plus inappropriée au fur et à mesure que le travail devient plus complexe et plus difficile à contrôler. L'expertise, par exemple, est un stock de travail accumulé qui produit son effet sans qu'il soit besoin pour cela d'un flux de travail : l'expert « voit » la solution du problème en un éclair, il ne lui reste plus qu'à la décrire et à la communiquer. La durée de travail est donc une mauvaise échelle de mesure pour évaluer la qualité d'une expertise – ou alors, il faudrait prendre non pas la durée d'une expertise, mais la durée d'accumulation du savoir de l'expert ; et il faudrait encore lui ajouter quelques paramètres qualitatifs. Or curieusement la plupart des tentatives de réponse aux questions énoncées aux premières lignes de ce chapitre, en tout cas dans le domaine de la gestion, se sont tournées vers des adaptations du modèle taylorien, pourtant déjà bien démodé, et entièrement fondé sur deux principes dont nous verrons dans la suite de cet ouvrage qu'ils sont particulièrement inadaptés à la construction de systèmes informatiques ou de logiciels :

— la séparation stricte entre travaux de conception et travaux d'exécution, confiés à des catégories de personnel fortement distinguées, typiquement les ingénieurs et les ouvriers de l'usine Ford des années 1920 ;
— le découpage des tâches d'exécution en gestes normalisés dotés de temps d'exécution fixes faciles à chronométrer.

Pour éclairer le sujet nous allons brosser un tableau rapide des procédures de contrôle du travail en général, avant d'aborder plus précisément leur application à l'informatique.

2.1 Le modèle de la grande industrie et le taylorisme

C'est au XVIII^e^ siècle que la vision du travail comme marchandise est vraiment devenue dominante, pour s'imposer au XIX^e^ siècle dans l'organisation type de la grande usine industrielle. Il n'est pas inutile de consacrer quelques lignes à ce modèle d'organisation, parce que même s'il est aujourd'hui en crise et en déclin, il existe encore beaucoup, et a même connu une vogue récente dans l'univers informatisé avec les centres d'appel téléphonique, qui ne sont rien d'autre que des bureaux taylorisés. Philippe Zarifian [133](p. 59), à qui ce chapitre doit beaucoup, caractérise cette organisation par la règle des trois unités du théâtre classique français :

— unité de lieu : les ouvriers sont réunis dans l'enceinte de l'usine où chacun a un poste assigné ;
— unité de temps marquée par les horaires collectifs, le pointage, et plus tard, avec Taylor, le chronométrage ;
— unité d'action, marquée par la cadence imposée, particulièrement nette pour le travail à la chaîne.

Zarifian rappelle combien il a été difficile, en France, d'imposer ce mode de travail à une population ouvrière issue en masse de la paysannerie indépendante et de l'artisanat. On signalera au passage le rôle qu'a joué le service militaire pour réduire l'individualisme des populations rurales et les plier à la discipline collective. Ce n'est que dans le second tiers du XX^e^ siècle que ce modèle se généralisera dans notre pays, bien après l'Angleterre et l'Allemagne.

Entre-temps était apparu le taylorisme, rationalisation ultime du système de la grande industrie. Zarifian le décrit comme « la soumission des actes de travail au calcul du temps, telle qu'elle détermine le choix imposé du contenu de ces actes (de ces gestes, dans le cas du travail ouvrier). » (p. 36). Curieusement, le but conscient poursuivi par Taylor en élaborant sa méthode était l'amélioration de la condition ouvrière. En fait, elle fut surtout un moyen de donner une apparence rationnelle au calcul de la valeur des biens produits par le travail.

Avec le taylorisme « la mesure normée du temps (le temps opératoire que l'ouvrier doit respecter, voire abaisser) s'incorpore dans les actes du travail.

Et le mot "incorporer" a un sens parfaitement précis : le temps pénètre dans les gestes et mouvements ouvriers, au point qu'à l'ouvrier échappe la définition du mouvement de son propre corps. Le mouvement de son corps lui est imposé comme une réalité à laquelle il doit se soumettre. Il y a, dans cette incorporation d'un temps abstrait au sein de l'usage de son corps, l'exercice d'une violence incommensurable, qui explique, en profondeur, pourquoi l'organisation taylorienne du travail était (est) destructrice des individualités, pourquoi elle a engendré de véritables révoltes… » (p. 37).

Pratiquement, le découpage du travail en tâches élémentaires dotées d'un temps d'exécution soigneusement calculé est insupportable pour celui qui doit l'effectuer, et cette méthode ne fonctionne que dans deux cas : si le travailleur s'impose à lui-même cette contrainte, parce qu'il y a un intérêt, quel qu'il soit, ou lorsqu'elle peut être appliquée par la force, ce qui peut être le cas du travail manuel simple, comme les coups de rame des galériens cadencés par les roulements d'un tambour, la progression des esclaves dans la plantation où le fouet

vient faire accélérer le retardataire, le travail dans la grande industrie contemporaine où la progression inexorable de la chaîne impose son rythme aux ouvriers spécialisés. Dans le cas d'un travail qui requiert l'initiative du travailleur, les moyens de mettre en échec la cadence prévue par l'encadrement sont innombrables, et comme la tentative de lui imposer cette organisation du travail a altéré la confiance du travailleur, il n'hésitera pas à y recourir à la moindre occasion. *A fortiori*, lorsqu'il s'agit d'un travail de conception, essayer d'imposer des normes de productivité de type taylorien ne peut que mener à la débandade ou, pire, au simulacre et à l'imitation de travail. Nous avons vu qu'Alexandre Zinoviev avait brillamment analysé le système bureaucratique soviétique dans cette perspective.

La discipline d'usine, parachevée par le taylorisme, participe de ce que Gilles Deleuze et Michel Foucault ont appelé une « société disciplinaire », dont la pratique s'élabore à partir du XVII^{e} siècle pour culminer au XX^{e}; ils sont ici évoqués par Philippe Zarifian : « dans ces sociétés, l'individu ne cesse de passer d'un milieu clos à un autre, chacun ayant ses lois : d'abord la famille, puis l'école, puis la caserne, puis l'usine, de temps à autre l'hôpital, éventuellement la prison... On sait que Foucault [...] a insisté sur le fait que le capitalisme industriel a largement emprunté à des modèles, dispositifs et savoirs qui s'étaient déjà constitués antérieurement, et dont l'asile et la prison fournissent le référent paradigmatique. » (pp. 13-14).

Après l'usine, le centre d'appel

Aujourd'hui le taylorisme au sens strict est en déclin parce qu'il n'est plus guère adapté aux besoins de la production industrielle contemporaine non plus qu'aux nouvelles normes de comportement individuel et collectif. La société disciplinaire cède le rang à la société de contrôle. Au chronométrage précis de chaque geste de l'ouvrier qui accomplit une tâche formalisée dans une table de temps, succèdent la fixation de délais et le contrôle informatique des activités des opérateurs des centres d'appel, dont il est facile de mesurer le temps moyen consacré à chaque appel de client et le nombre d'appels pris. De tels centres d'appel peuvent être installés dans des pays francophones à coût de main-d'œuvre modéré, ainsi il en existe un à Tunis qui a pour clients simultanés des acteurs majeurs du secteur français des télécommunications, de l'Internet et du câble. Les opératrices doivent pour être sélectionnées posséder une parfaite maîtrise de la langue française et la parler sans accent, ce qui suppose un niveau d'instruction élevé. Il leur est demandé de se présenter aux clients sous des pseudonymes tels que Sandrine, Annabelle ou Virginie. Leurs salaires sont bien entendu une faible fraction du SMIC français. Ces centres d'appel sont utilisés pour répondre aux demandes de renseignements ou d'assistance technique *(hotline)* des clients, mais aussi pour harceler au téléphone les clients potentiels.

Un dispositif aussi artificiel ne peut pas donner de bons résultats : les univers des deux interlocuteurs au bout du fil sont tellement éloignés l'un de l'autre que les réponses aux questions ne peuvent être que stéréotypées, et dès que la question se complique un peu le dialogue est impossible – pour être juste, il convient de dire que la qualité de réponse des centres d'appel installés en France n'est pas meilleure. Les services d'assistance téléphonique *(hotline)* des grandes entreprises de télécommunication et d'informatique mis en œuvre par de tels procédés recueillent un taux de satisfaction des utilisateurs très bas, ce qui semble ne troubler personne : une fois que le client est abonné il est considéré comme captif, notamment parce que les tarifs sont conçus de telle façon que le changement de fournisseur a un coût élevé.

Ce qui a séduit les managers dans l'idée de centre d'appel, c'est bien sûr l'aspect taylorien. Pourquoi ce système ne peut pas marcher, ce sera l'objet de la section suivante.

Incidemment nous pouvons vérifier, par exemple dans le cas concret du groupe Bouygues [20], ce que nous écrivions au chapitre précédent : les entreprises qui échappent au diktat des bureaucrates financiers agissent différemment : sur 6800 salariés, Bouygues

Télécom compte 2300 conseillers de clientèle, qui travaillent en fait dans des centres d'appel, mais dont le sort est bien différent de ce qui est décrit ci-dessus. Fait déjà remarquable, ils sont des salariés de l'entreprise, et non d'une quelconque société d'intérim, et même pour certains d'entre eux ils travaillent dans les mêmes locaux que la direction générale. En outre, leur employeur s'évertue à recueillir et à exploiter le savoir qu'ils accumulent à l'écoute des clients, au lieu de simplement l'ignorer comme cela a lieu le plus souvent. On pourra consulter à ce sujet le site de Michel Volle [126]. On pourra aussi voir avec profit le film d'Arnaud et Jean-Marie Larrieu, « Un homme, un vrai », avec Hélène Fillières et Mathieu Amalric, qui se passe dans l'univers grisant des managers de centres d'appel. Pour mémoire, les centres d'appel emploient en France 210 000 personnes au 1er janvier 2005, pour 1 million en Grande-Bretagne.

2.2 Tout travail émet de la pensée

Le travail a vocation à produire du sens, pour son auteur comme pour son destinataire. La production d'un bien ou d'un service a du sens si elle a valeur d'usage pour une clientèle ou un public. Le travailleur qui les produit en a conscience et cette conscience guide ses actes. Il appartient aux organisateurs du travail de veiller à ce que cette production de sens ait lieu, sauf à vouloir régner sur un univers tel que décrit dans les films « Metropolis » de Fritz Lang ou « Les temps modernes » de Charlie Chaplin, ou encore dans les pièces de Georges Courteline (on rêve à ce que Courteline aurait pu imaginer s'il avait connu les systèmes d'information et les projets !). Suivons ici encore Philippe Zarifian qui nous invite à rendre visite à Gilles Deleuze et à Michel Foucault : « travailler, c'est s'affronter à des situations qui comportent du surprenant, de l'inédit, de l'imprévu, et c'est, pour employer un concept de Gilles Deleuze, *contre-effectuer ces événements,* leur conférer un sens humain et agir en conséquence...

À tout moment, un individu au travail peut perdre le sens de ce qu'il fait et tomber dans un état d'aliénation. La pression du débit, l'impossibilité de comprendre les attendus et finalités de ce qu'il fait, l'opacité de l'organisation dans laquelle il est placé... peuvent engendrer une forte difficulté à produire du sens. L'aliénation... surgit par *perte de sens*, enfermement dans une pure routine reproductrice... et dans un champ de pressions que l'on ne peut que subir. » (p. 9-11).

La réalisation d'un travail, comme l'explique encore Philippe Zarifian [133] (p. 83), passe par deux moments au moins : un moment de pensée, où l'on imagine le résultat à venir d'un travail futur et les avantages qui pourraient en résulter, un moment de réalisation de la chose projetée. La division du travail (et plus particulièrement l'organisation taylorienne du travail) vise à ce que celui ou ceux qui pensent ne soient pas les mêmes que celui ou ceux qui réalisent, mais il n'est pas si facile de supprimer toute pensée dans la phase de réalisation. Et cela dépend beaucoup de la nature du travail.

Dans le bâtiment, ces deux moments du travail sont instanciés en deux personnes (généralement collectives) : le maître d'ouvrage, qui dit ce qu'il faut faire (et paie), et le maître d'œuvre, qui organisera la réalisation, avec, entre les deux, l'architecte qui dit comment ce sera réalisé. Notons au passage le caractère discursif essentiel du travail de définition : il faut des échanges verbaux entre ces différentes personnes pour que le projet aboutisse.

Évidemment le résultat final sera différent de ce qui était prévu par le maître d'ouvrage et par le maître d'œuvre, parce que le travail de réalisation soulèvera des problèmes imprévus et imprévisibles, qui obligeront tous les participants à apprendre et à imaginer des choses nouvelles pour les résoudre.

Retenons que la bonne fin de la réalisation d'un projet architectural de quelque ampleur suppose une véritable collaboration entre le maître d'ouvrage, l'architecte et le maître d'œuvre, c'est-à-dire que les problèmes de réalisation rencontrés par la maîtrise d'œuvre

doivent pouvoir être résolus par une évolution du projet prise en compte par la maîtrise d'ouvrage et l'architecte, en d'autres termes la réalisation doit pouvoir agir en retour sur la conception. Pour les projets informatiques nous verrons qu'il en va de même, et que si cette possibilité d'effet rétroactif est bloquée, par exemple par un dispositif aberrant comme le Code des Marchés Publics que nous évoquerons à la section suivante, eh bien le projet échoue, peu ou prou.

2.3 Théorie et pratique de la commande publique

En France, les prestations de services commandées par les services publics à des entreprises font l'objet de contrôles de leur bonne réalisation selon des procédures et des règles qui sont des cas particuliers d'un ensemble plus vaste, la réglementation des marchés publics de l'État, dont nous allons donner ci-dessous une brève description.

2.3.1 Réglementation des marchés publics

Le dispositif juridique, réglementaire et comptable qui encadre les actes contractuels de la puissance publique en France est très particulier et il convient d'en dire quelques mots. Hormis d'anciennes colonies françaises auxquelles nous l'avons transmis, aucun pays au monde n'est doté d'un système aussi étrange. Dans la plupart des pays, on considère en effet que les lois et les règlements qui s'appliquent au commun des mortels peuvent s'appliquer aux services publics, à quelques ajustements techniques près : idée totalement inappropriée aux yeux de la puissance publique française ! Le Service public français est doté d'un système juridique parfaitement extravagant du droit commun[1].

Si l'on veut comprendre quelque chose à ce dispositif (et toute entreprise désireuse de vendre des biens ou des services à des administrations publiques ou locales devra en comprendre au moins les grandes lignes, sauf à subir de graves mésaventures qui pourront aller, le cas n'est pas si rare, jusqu'au dépôt de bilan), il faut bien dès l'abord saisir sa complexité, parce que si chacun de ses éléments pris isolément peut sembler plein de bon sens, ce n'est que leur combinaison, conjuguée à quelques traditions non réglementaires dans leur mise en œuvre, qui en fait un système bureaucratique absurde et inextricable tel que ceux si bien évoqués par Georges Courteline.

Premier principe : séparation de l'ordonnateur et du comptable

Le premier élément du dispositif est le principe de séparation de l'ordonnateur et du comptable. Il a été instauré en 1319 par l'ordonnance portant création de la Cour des Aides, promulguée au Vivier-en-Brie par Philippe V le Long (1317 – 1322). Le roi, soucieux d'éviter les détournements de fonds, décidait que celui de ses fonctionnaires qui prenait l'initiative d'une dépense (l'ordonnateur) ne pourrait pas être celui qui payait effectivement (le comptable). Ce principe se traduit aujourd'hui par le fait que si le directeur de tel organisme public de recherche (l'ordonnateur) décide d'effectuer telle dépense, pour laquelle le Parlement a voté un budget, celui qui paiera sera l'Agent Comptable de l'établissement, qui n'est pas son subordonné hiérarchique (il rend compte au directeur du Budget du ministère des Finances) et qui peut refuser de payer (il ne s'en prive pas).

1 Que dire de la justice administrative ! Il est permis de s'interroger, et sur le sens de la locution, et sur le bien-fondé de l'instance qu'elle désigne. Ne serait-elle pas à la justice ce que la musique militaire est à la musique (encore que cette dernière ne soit pas dépourvue de qualités...). L'administration ne pourrait-elle répondre aux mêmes tribunaux que les simples citoyens ?

Second principe : contrôle *a priori*

Le second élément du dispositif est le principe du contrôle *a priori*. Lorsque le directeur de l'organisme public de recherche pris ici comme exemple (l'ordonnateur) décide d'engager une dépense, il n'envoie pas au fournisseur un bon de commande qui engagerait l'établissement : il le soumet au Contrôleur financier, qui dépend hiérarchiquement du directeur du Budget du ministère des Finances, et dont seul le contre-seing donnera une valeur d'engagement légal au bon de commande. Le contrôle *a priori* est sans doute l'élément le moins justifiable du dispositif : les directeurs des établissements publics de recherche tels que le CNRS ou l'Inserm sont nommés par le Président de la République, en Conseil des Ministres, ce qui leur confère une légitimité émanée assez directement du peuple souverain ; que leur signature, pour engager l'établissement qu'ils dirigent, doive être contresignée par un obscur fonctionnaire désigné selon un processus bureaucratique opaque, est un déni infligé à la démocratie censée gouverner notre pays. Ce principe a une conséquence délétère très lourde : tous les fonctionnaires, à l'exception des contrôleurs financiers, des agents comptables et de leurs consorts, sont considérés comme des mineurs irresponsables. Lorsque l'on considère les gens comme des irresponsables, ils ont tendance à se comporter de façon irresponsable : ce phénomène n'est pas absent de la fonction publique française.

Rappelons que des millions d'entreprises en France et dans le monde ont recours, pour vérifier que leur argent n'est pas détourné, à des procédures de contrôle *a posteriori*, et rien ne prouve que le système public français soit une protection plus efficace contre les malversations : si tel était le cas, la France obtiendrait sans doute un meilleur classement sur l'échelle de l'indice de perception de la corruption établi par l'institut *Transparency International* [112].

Les deux premiers principes de la commande publique sont mis en œuvre conformément aux dispositions du décret du 29 décembre 1962 portant règlement général sur la comptabilité publique, assorties d'innombrables autres textes qui règlent différents détails. Peu de textes sont abrogés, de nouvelles règles viennent seulement s'ajouter à celles qui existent déjà.

Troisième principe : le Code des Marchés Publics

Le troisième pilier de la commande publique est le Code des Marchés Publics (CMP), qui régit tous les contrats, conclus par des organismes publics ou des collectivités territoriales, dont le montant excède un certain seuil (au jour d'écriture de ces lignes : 150 000 Euros HT pour l'État et 230 000 Euros HT pour les collectivités territoriales ; on consultera avec profit à ce sujet le site du Ministère de l'Économie et des Finances [83], où l'on trouvera notamment le texte du Code lui-même). Ce code semble à première vue plein de bonnes idées, destinées notamment à protéger l'acheteur contre les clauses abusives que les fournisseurs ne se privent guère de glisser dans les contrats-type qu'ils imposent plus ou moins à leurs clients privés. En fait, ce qui fait que ce texte devient assez vite un obstacle plutôt qu'une aide, ce n'est pas tellement sa longueur (69 pages) mais sa complexité, accrue par les correspondances croisées qui le combinent avec d'autres textes (au nombre de 33 dans la version du 7 janvier 2004, disponible en ligne) que le lecteur est censé connaître, et bien sûr avec les deux premiers Principes, surtout le second (contrôle *a priori*). Le cadre imposé aux parties est formel, rigide, inspiré par la défiance de l'instance de contrôle envers les parties contractantes, et des parties contractantes l'une à l'égard de l'autre (article 64 : « Il ne peut y avoir de négociation avec les candidats. ») Bref, lorsque l'on entame une procédure régie par le Code des Marchés Publics, il faut s'attendre à neuf mois de formalités pour un cas simple et un montant modéré. Le poids d'une telle procédure est tout simplement insupportable.

Cette lourdeur du Code des Marchés Publics a des effets pervers. Ainsi, dans le cas d'un marché de prestations intellectuelles, par exemple la réalisation d'un système informatique

comptable et financier, le Code prévoit des clauses perfectionnées qui permettent la résiliation aux torts du titulaire du marché s'il ne remplit pas sa tâche de façon satisfaisante. Mais le client serait bien en peine de les appliquer, parce qu'il lui faudrait alors lancer une nouvelle consultation pour choisir un autre fournisseur, procédure qui aboutirait au bout de neuf mois si tout se passe bien – n'oublions pas qu'à tout moment le contrôleur financier peut bloquer toute l'opération *sine die* : comment assurer la continuité du service pendant ce délai ? Le Code des Marchés Publics, censé protéger le client, le livre en fait pieds et poings liés à son fournisseur défaillant.

Il y a aussi dans le Code quelques attrape-nigauds : il comporte ainsi un alléchant article 37 consacré aux « marchés de conception-réalisation ». Ce type de marché semble parfaitement adapté au processus de collaboration dans la durée qui doit régler les relations entre maîtrise d'ouvrage et maîtrise d'œuvre dans la conduite d'un projet informatique : il concerne « les marchés qui portent à la fois sur la définition du projet et sur l'exécution des travaux... Sont concernés des ouvrages dont la finalité majeure est une production dont le processus conditionne la conception et la réalisation, ainsi que des ouvrages dont les caractéristiques, telles que des dimensions exceptionnelles ou des difficultés techniques particulières, exigent de faire appel aux moyens et à la technicité propres des entreprises. » Mais il ne faut pas rêver, les contrôleurs veillent et disposent d'une batterie d'arguments pour rejeter toute tentative de recours à ce dispositif, sauf cas très particuliers, ce qui est un exemple de détournement de la volonté du législateur.

L'article 36 est consacré à une procédure dite de « dialogue compétitif », censée faciliter la définition d'un cahier des charges, qui risque fort d'être également décevante, compte tenu des habitudes séculaires des administrations financières.

En trois décennies de pratique des marchés publics, l'auteur de ces lignes a vu paraître une demi-douzaine de nouvelles versions du Code ; chacune était censée alléger les procédures et permettre enfin de travailler normalement, c'est-à-dire sur la base d'une confiance réciproque entre client et fournisseur, établie par des moyens normaux (expérience, réputation, échanges d'expérience avec d'autres organismes), et d'un contrôle *a posteriori* des actes accomplis. En réalité chacune de ces nouvelles versions du Code était aussi indigeste que les précédentes, avec l'inconvénient supplémentaire d'invalider le capital de compétence acquis pour l'utiliser et d'obliger à un effort significatif d'adaptation aux nouvelles règles. Les deux dernières versions, sous l'influence de la loi n° 93-122 du 29 janvier 1993 relative à la prévention de la corruption et à la transparence de la vie économique et des procédures publiques, dite loi Sapin [73], dont pourtant certains articles ont été déclarés anticonstitutionnels par le Conseil Constitutionnel, imputent aux Personnes Responsables des Marchés une responsabilité exorbitante, puisque si elle n'accroît que modérément leur capacité d'initiative, elle fait peser sur leurs têtes une menace judiciaire très lourde. En effet tout soupçon de partialité dans la rédaction d'un appel d'offres peut les conduire devant une juridiction pénale. En fait, tout maître d'ouvrage compétent a une idée de la façon dont un projet doit être réalisé, ce qui le place sous le coup de la loi, risque que n'encourt pas l'incompétent total qui écrit n'importe quoi sans rien savoir, attitude qui, on s'en doute, a les faveurs des instances de contrôle (grâce au ciel, l'incompétence totale n'est pas une denrée introuvable dans l'univers administratif).

Un autre effet pervers de la rigidité du Code des Marchés Publics, conjuguée avec les interventions imprévisibles et bloquantes des agents comptables et des contrôleurs financiers, c'est que l'acheteur public est sûr de se retrouver un jour ou l'autre dans une situation où il sera dans son tort à l'égard du fournisseur, ne serait-ce que parce qu'une fourniture livrée sera payée avec retard : ce genre de situation met l'acheteur public en position de faiblesse dans tout contentieux avec son fournisseur, ce qui, *a priori*, n'est pas l'objectif visé.

La rigidité de ce cadre réglementaire constitue sa principale faiblesse : comme il rend à peu près impossible toute collaboration réelle entre administration et fournisseur, il engendre lui-même les procédés qui permettent de le contourner, et une fois que ceux-

ci sont maîtrisés, ils peuvent être appliqués à des fins que la loi et la morale réprouvent. S'il était un obstacle sérieux aux fraudes et aux malversations, les journaux seraient moins pleins de leur récit. Lors d'un entretien publié par la Revue des Secrétaires Généraux des universités [28], Jérôme Grand d'Esnon, Directeur des Affaires Juridiques au Ministère de l'Économie et des Finances, ne peut que constater la coïncidence géographique entre le territoire des marchés publics régis par une réglementation très rigide, et le territoire de l'occurrence maximum de la corruption en matière de marchés publics :

« La tradition gréco-latine avait imposé une solution de type procédural (appel d'offres + cloisonnement, c'est-à-dire interdiction de dialogue et d'échange). La tradition nordique, anglo-saxonne, est plus ouverte sur les pratiques de la négociation. Or, si la solution latine est bonne en situation d'achat simple, elle est stupide et dangereuse en situation d'achat complexe (celle où l'acheteur ne sait pas rédiger lui-même son cahier des charges ; celle où il a besoin de lisibilité ; celle où il a besoin de pouvoir dialoguer avec ce que lui offre le marché, ce que proscrit la procédure). Or, on peut relever que si l'on superpose la carte des conceptions culturelles en matière de marché public (zone Sud France, Italie, Grèce – pour l'appel d'offres et le cloisonnement ; zone Nord pour la négociation) et celle de la corruption en matière de marchés publics, on constate un recouvrement quasi-parfait. »

La LOLF est-elle un espoir ?

La réglementation qui s'applique aux marchés publics est elle-même enchâssée dans un cadre plus ample : la *loi organique relative aux lois de finances* (LOLF), qui prétend représenter la constitution financière de la République. L'avant-dernier avatar de cette réglementation était l'Ordonnance de 1959, qui avait résisté victorieusement à 36 tentatives de réforme par le Parlement. De façon un peu inattendue, le Parlement a voté le 1er août 2001 la *Loi organique relative aux lois de finances* (LOLF) nº 2001-692 publiée au Journal Officiel du 2 août 2001. Ce texte a réuni de façon surprenante les votes des partis politiques de toutes tendances, le soutien du Président de la République et l'accord du Conseil Constitutionnel. La teneur de ce texte est très novatrice.

Les deux axes de la LOLF sont le passage d'une administration des moyens à une administration de résultats, et le renforcement du pouvoir de contrôle du Parlement, assuré par une transparence accrue des finances publiques destinée à mettre fin à la situation présente, où le parlement vote sans vraiment en juger une Loi de Finances inintelligible au possible. Il est question de clarté des objectifs, et de mesure des résultats. Les gestionnaires deviendraient responsables, leur comptabilité serait soumise aux mêmes règles que celle des entreprises, le contrôle *a priori* céderait la place au contrôle *a posteriori*.

Tout ceci semble bel et bon, mais il est permis d'émettre quelques doutes quant aux effets réels de ce programme. Ces doutes sont nourris d'abord par l'expérience des réformes antérieures, certes moins ambitieuses, mais qui affichaient des objectifs analogues sans jamais les atteindre. De plus, la complexité même de la loi, qui ne manquera pas, tel un porte-avions amiral, d'être escortée de toute une flottille de lois annexes, de décrets d'application et de règlements divers et variés, laisse planer un certain scepticisme sur ses bienfaits. Enfin il faut bien voir que les principes de la LOLF sont peu compatibles avec le statut de la fonction publique : soit les fonctionnaires sont responsables, soit il faut les contrôler étroitement, or leur statut actuel ne permet pas qu'ils soient responsables. La seule chose sûre, c'est qu'elle va susciter une grande agitation administrative et des revenus élevés pour les sociétés de services informatiques qui vont réaliser les logiciels comptables et financiers nouveaux requis par ses dispositions.

2.3.2 La pratique des marchés publics

Lorsque l'administration française fait réaliser un système informatique par un prestataire, elle est en position de maître d'ouvrage. Elle rédige (ou fait rédiger) un cahier des charges qui décrit les spécifications du système à réaliser. Ce cahier des charges constitue la partie technique des documents du marché fournis aux entreprises candidates lors de la consultation d'appel d'offres. À l'issue de cette consultation le marché sera attribué au prestataire qui aura fourni la meilleure réponse. La réglementation des marchés publics qui s'impose à ce processus est très rigide, elle limite notamment très rigoureusement toutes les conversations entre maître d'ouvrage et maître d'œuvre préalables aux travaux, et rend très difficile tout ajustement des clauses du marché aux événements imprévus provoqués par le déroulement de la réalisation. De tels événements sont d'ailleurs considérés comme des failles du dispositif contractuel : le maître d'ouvrage aurait dû les prévoir.

Cette vision de l'Administration laisse planer un doute sur sa capacité à concevoir même ce qu'est le travail, et de là à savoir le respecter, puisqu'elle n'admet pas que puissent surgir des problèmes inédits susceptibles de solutions nouvelles imprévisibles. Cette conception assez mortifère (ce qui est parfaitement prévisible c'est ce qui est mort) s'appelle la *bureaucratie*, et ne semble pas étrangère au taux de succès assez faible des projets de l'Administration dans le domaine informatique.

Plus précisément, faire fonctionner un dispositif du monde réel, tel que téléphérique, hélicoptère ou ordinateur, impose des contraintes opérationnelles fortes qui sont incompatibles avec le Code des Marchés Publics et le Règlement de la Comptabilité Publique. Pour contourner ces réglementations d'un autre âge, pratiquement incompatibles avec un travail réel, il est tentant (et très couramment pratiqué, notamment dans le monde de la recherche) de recourir à la création de sociétés civiles ou d'associations régies par la loi de 1901. Il est à craindre que parfois cela ne fasse qu'accroître la gabegie et l'impéritie.

En cette année 2004, un ancien secrétaire d'État est mis en examen devant la Cour de Justice de la République pour détournement d'argent public au moyen d'une association, créée à cet effet, qui recevait des subventions et rémunérait grâce à elles des personnels de statut privé. À ce compte, des dizaines de directeurs d'unités de recherche rattachées à des organismes publics devraient être traduits en justice, ainsi que les agences et associations qui les soutiennent. Un chercheur soucieux de faire de la recherche avec efficacité ne peut en effet s'en remettre pour le fonctionnement de son équipe aux délais imprévisibles et aux méandres de procédure courtelinesques d'un organisme public : il créera donc une association, au compte de laquelle il fera verser ses subventions de recherche. Il est indéniable qu'une fois qu'un tel dispositif a été mis en place pour échapper à une réglementation inapplicable, il peut être utilisé pour le meilleur comme pour le pire, pour la recherche comme pour le détournement. Quant aux agences de financement, elles versent des bourses qui permettent de rémunérer, en dehors de tout système de protection sociale, des personnels qui n'ont pas trouvé place dans les corps statutaires – bref, c'est du travail au noir, encore plus que dans le cas des personnels temporaires employés par l'administration sous le nom de vacataires, un non-statut qui conduirait rapidement devant les tribunaux le dirigeant d'entreprise qui aurait l'idée de s'en inspirer pour ses propres salariés.

Dans le domaine du bâtiment et des travaux publics, les liens traditionnels et cordiaux qui associent en France les entrepreneurs et les notables et politiciens locaux permettent d'aplanir bien des difficultés administratives. Il faut dire que les élus locaux, qui ont besoin d'obtenir des résultats tangibles pour être réélus dans un délai de cinq ou six ans, n'entendent pas se laisser mettre des bâtons dans les roues par les membres de la fonction publique territoriale, et qu'ils disposent de moyens de pression plus efficaces que ceux des chercheurs.

Pour nous résumer, le dispositif français des marchés publics est une calamité. Dans un contexte où plus de la moitié du PIB passe par le budget de l'État d'une façon ou d'une autre un système aussi anachronique et inefficace constitue un boulet que la nation traîne à

son pied ; il est dans ces conditions miraculeux que la France figure encore au nombre des cinq ou six pays les plus riches de la planète, et cela donne une idée des efforts considérables consentis par les entreprises privées et leurs salariés pour surmonter un tel handicap.

données 2016, source IFRAP	milliards €
PIB français	2229
marchés publics	334
rémunération des fonctionnaires	282
Total des dépenses des administrations	1253

Trouver sur les sites de l'Insee ou du Ministère de l'Économie des informations sur la dépense publique ou sur la fonction publique relève du rébus, mieux vaut un moteur de recherche.

2.3.3 Quels sont les services publics « rentables » ?

Pour parler comme les informaticiens, nous pouvons identifier un « effet de bord », c'est-à-dire une conséquence non intentionnelle de la réglementation des marchés publics : les administrations ne disposent d'aucun moyen pour envisager la notion d'investissement. Le connaisseur des comptes publics aura sans doute sursauté à cette assertion : il faut donc que je précise le propos. Effectivement, les plans comptables utilisés par les services publics comportent bien des comptes d'investissement. Ce qui en est absent, c'est le moyen de rapprocher une rentrée d'argent d'une dépense, pour associer par exemple un investissement à la ressource qui l'a autorisé, ou aux ressources ultérieures qu'il a permis de réunir. Je n'ignore pas que cet état de fait a des raisons : il ne sied pas que la façon dont l'État collecte tel impôt soit associée de façon obligée à tel type de dépense. Simplement, la comptabilité publique est conçue plus pour le contrôle de la conformité des opérations aux règlements que pour l'efficacité de la gestion, et cette façon de tenir des comptes a des inconvénients, notamment elle ne favorise pas la bonne appréhension des investissements. Or toute une partie de l'enjeu qui est en cause pour savoir si l'informatisation va réussir ou échouer, c'est de savoir si on va la considérer comme une charge (et alors on échouera), ou comme un comme un investissement, ce qui lui donnera quelques chances de succès.

Pendant les deux années que j'ai passées au sein de l'administration centrale du Ministère de l'Économie et des Finances, j'ai observé un effet curieux mais à mon avis pervers et gravissime de cette vision : la Direction Générale des Impôts, la Direction de la Comptabilité Publique et la Direction Générale des Douanes et des Droits Indirects étaient considérés comme les services « rentables » de l'administration, par opposition aux services « dépensiers » comme la Santé, l'Éducation nationale, etc. C'est un peu comme si, chez Renault, on considérait que le service comptable était celui qui faisait vivre la société. En fait, si les contribuables français acceptent bon gré mal gré de payer leurs impôts sans trop égorger de percepteurs, c'est bien parce qu'une pratique démocratique de longue durée leur a enseigné que cet argent qu'ils ont du mal à payer leur reviendra un jour sous la forme d'écoles pour leurs enfants, de routes et d'hôpitaux, de police et de défense nationale. Les administrations financières ne sont dans tout ceci qu'un rouage utilitaire subalterne, dont le coût devrait être allégé au maximum, alors qu'elles sont les mieux traitées, en dotation budgétaire comme en primes pour le personnel. C'est assez aberrant, pour rester sobre.

2.4 Projet et cahier des charges

Jean-Pierre Boutinet [19] nous guidera ici pour ce qui concerne l'histoire de la notion de projet. Ce terme apparaît d'abord dans le contexte architectural pour désigner un élément de construction situé en avant de la façade, mais le premier projet au sens moderne est dû à Filippo Brunelleschi, créateur du *Duomo* de Florence, Santa Maria del Fiore, au début du

XV^e^ siècle. Auparavant les constructions de grande ampleur étaient réalisées par un concours pas toujours très ordonné de maçons, de charpentiers, de couvreurs, de sculpteurs, etc., sous la direction d'un architecte. On peut certes déjà identifier de grands maîtres d'ouvrage, comme l'abbé Suger pour la basilique de Saint-Denis, ou de grands architectes comme Villard de Honnecourt, mais Brunelleschi a voulu une démarche plus formalisée, au moyen d'un document écrit et dessiné préalable à toute exécution afin de limiter l'autonomie des corps de métier impliqués dans la réalisation[2]. Ce document sera nommé le *cahier des charges* du projet. Il semble que ce soit grâce à cette démarche centralisée et rationnelle et à la division du travail qui en découlait qu'ait été rendue possible la prouesse architecturale constituée par la coupole de la cathédrale de Florence.

Nous retrouverons ces notions de projet et de cahier des charges parmi les méthodes envisagées pour mener à bien la réalisation de systèmes informatiques.

Il est clair qu'il semble judicieux, avant de commencer un travail de quelque ampleur, de savoir le mieux possible ce que l'on veut faire. Le consigner par écrit ne peut qu'aider, surtout si le travail doit ensuite être réalisé par plusieurs personnes, éventuellement liées les unes aux autres par des contrats. Vérifier à intervalles réguliers par des procédures elles-mêmes écrites que ce qui est fait correspond bien à ce qui avait été décidé ne devrait que contribuer au succès.

Jean-Pierre Boutinet nous indique par ailleurs ceci : « La gestion par projet se veut être un mode original de gouvernement qui vise à déterminer les meilleures conditions dans l'implantation d'une innovation au sein d'un ensemble organisationnel, qu'il s'agisse d'une innovation technologique, d'une innovation comptable, d'une innovation sociale … Au lieu de faire transiter l'innovation en cause par la hiérarchie, on la confie directement à une équipe autonome, qui aura la plus large latitude pour intégrer cette innovation aux secteurs concernés de l'entreprise. » Et Juliette Poupard [**poupard**] ajoute : « Ainsi l'organisation par projet est-elle censée dépasser les limites de l'organisation pyramidale classique et proposer des dispositifs de travail propres à encourager la coopération transversale entre les acteurs. » Comment ne pas approuver un programme aussi intelligent ?

Alors pourquoi cette démarche en apparence si rationnelle et sage échoue-elle si souvent à éviter l'échec dans le domaine de l'informatique de gestion ?

Avant d'entrer dans le vif de ce sujet, rappelons que nous avons réparti les systèmes informatiques en trois classes, ceux qui sont destinés à faire fonctionner les systèmes d'information des organisations (classe 1), ceux qui assurent la commande et le contrôle des systèmes techniques (classe 2) et ceux de la classe 3 qui constituent des infrastructures informatiques (voir page 6). Le présent ouvrage est essentiellement consacré aux systèmes de la première classe, et il n'y sera question de ceux des autres classes qu'incidemment.

Lois physiques pour systèmes informatiques

Il convient de souligner une caractéristique des systèmes informatiques de classe 2, qui les distingue des systèmes de classe 1 : un vaisseau spatial, un robot industriel ou un véhicule terrestre obéissent aux lois de la physique, dont personne n'a dit qu'elles étaient simples, mais qui ont le mérite d'avoir été étudiées depuis des siècles, ce qui a permis de les doter de formalisations mathématiques rigoureuses. Les lois de la physique procurent aux systèmes de classe 2 (souvent dits *embarqués*) un système de références auquel se reporter pour savoir si un algorithme est correct. Il n'existe rien de tel pour les systèmes de classe 1, destinés aux Systèmes d'information (SI), qui opèrent sur des données issues de sociétés humaines, pour lesquelles n'existe rien de comparable aux lois de la physique, heureusement d'ailleurs (tous les totalitaires en rêvent).

2 Il est possible de se faire une idée exacte de sa conception en visitant à Florence le Musée des Travaux du Duomo, qui expose maquettes, dessins et autres documents.

2.4.1 La frontière entre conception et fabrication

La vision classique de la conduite d'un projet informatique de gestion est la suivante : le maître d'ouvrage élabore (ou fait élaborer) un cahier des charges, complété par un document de spécifications détaillées qui décrit par le menu les fonctions qui devront être assurées par le logiciel à réaliser. Avec le bon à tirer des spécifications détaillées se clôt la phase de conception. Le maître d'œuvre, choisi à l'issue d'un appel d'offres pour la qualité de sa réponse au cahier des charges, entreprend la phase de fabrication du logiciel conformément aux spécifications détaillées. Il suffit, à chaque étape du travail, de vérifier la conformité aux spécifications pour que tout se passe bien, et qu'à la fin le logiciel livré soit satisfaisant. Cette vision, bien résumée par le schéma de la figure 2.1Cycle de développement « en V » du logiciel selon la vieille écolefigure.2.1, est particulièrement ancrée dans l'administration, et de façon générale elle plaît aux managers qui pensent parfois qu'une incompétence technique distinguée, en les plaçant « au-dessus de la mêlée », leur évite les partis-pris qui seraient le lot des simples ingénieurs. Eh bien la technique se venge : la démarche que nous venons de décrire échoue, et ne peut qu'échouer, parce qu'elle repose sur une erreur.

Quelle est cette erreur ? C'est de croire que les spécifications détaillées constituent la conception, et que la rédaction du texte des programmes dont se composera le logiciel constitue la fabrication. Comme l'ont bien mis en lumière les auteurs du livre « L'*eXtreme Programming* » [9], la rédaction du texte des programmes, en un mot la programmation, appartient intégralement au travail de conception. Ce qui correspond à la fabrication, c'est la traduction des programmes en langage machine et leur assemblage, opérations réalisées par des logiciels. Le chapitre 4 apportera des précisions à ce sujet.

Dès lors que la programmation est reconnue comme une activité complexe qui demande de la créativité, il est clair qu'elle va soulever des problèmes inédits qui vont nécessiter de revenir sur les spécifications pour les modifier, et donc que le scénario proposé ci-dessus ne peut pas aboutir au succès. À plus forte raison, si la programmation change de statut pour passer de la phase de fabrication (sous-entendu, de travail simple facile à quantifier) à la phase de conception, toute méthode qui repose sur le scénario en question est vouée à l'échec. Et si de plus le projet est encadré par un règlement tel que le Code des Marchés Publics, qui interdit toute discussion entre les auteurs du cahier des charges et les responsables de la programmation avant la remise du cahier des charges, qui ensuite est figé contractuellement, les dernières chances de succès s'évanouissent à l'horizon, d'autant plus qu'au fil des deux ou trois années que dure la réalisation d'un projet informatique les desiderata de l'utilisateur final auront de toute façon changé – souvent à un point tel d'ailleurs que le projet formulé initialement n'a plus aucun intérêt pour lui – et que le scénario n'envisage aucune façon d'en tenir compte. Hélas pour le contribuable...

2.4.2 Bâtiment, mécanique, programmation : le démon de l'analogie

Nous y reviendrons au chapitre 4, mais nous savons déjà que la mise en œuvre de l'informatique s'est beaucoup inspirée des procédures de travail les plus élaborées du XX^{e} siècle, qui étaient celles de l'industrie mécanique. Celle-ci disposait d'un outil de conception, le dessin industriel, qui permettait la séparation entre conception et fabrication tout en assurant la transmission précise des spécifications et des consignes de l'une à l'autre. L'informatique a toujours cru manquer d'un tel outil de conception, et a fébrilement cherché à s'en doter, de la méthode Merise à UML *(Unified Modeling Language)*. En fait, elle possède des outils de conception, sous la forme des langages de programmation, mais ils semblent de trop bas niveau aux yeux de certains maîtres d'ouvrage, alors que si la programmation des ordinateurs s'avère plus difficile, beaucoup plus difficile que le dessin industriel, c'est tout simplement parce que la spécification d'un logiciel est beaucoup plus complexe que celle d'un appareil mécanique.

Quelques années après l'analogie entre l'industrie mécanique et l'informatique, la métaphore architecturale est apparue. Il faut reconnaître qu'elle n'est pas dépourvue de séductions. Pour l'architecte qui dessine un bâtiment, le plus important, ce ne sont pas les pièces, mais les moyens de circuler entre elles : escaliers, couloirs, vestibules et patios, munis des portes et fenêtres qui conviennent. De même pour l'urbaniste ce sont les rues, places, ronds-points, ponts et tunnels qui organisent un espace où il plantera les immeubles. Cette façon de voir les choses n'est pas sans intérêt pour l'informaticien, qui sera bien avisé de s'en inspirer pour organiser la division de son programme en sous-programmes en tenant compte des données qu'ils auront à s'échanger. Le chapitre 4 traitera ce sujet. L'analogie avec le bâtiment permet aussi d'introduire le couple maître d'ouvrage – maître d'œuvre, en oubliant curieusement l'architecte au passage, et de renforcer le rôle du cahier des charges.

Finalement ces deux analogies ont surtout engendré des erreurs. Un objet mécanique, comme un bâtiment, est un objet physique que l'on peut dessiner (que ce soit avec un crayon ou avec un ordinateur) de façon précise. Le dessin pourra être montré au maître d'ouvrage, pour qui l'on pourra aussi fabriquer des maquettes. J'ai eu un professeur de dessin industriel qui nous parlait avec mépris du dessin en perspective, destiné selon lui aux mécaniciens, incapables à l'en croire de comprendre un vrai dessin selon les trois axes. Incidemment, ma courte expérience du dessin industriel (le vrai, selon trois axes) m'a suggéré que le processus d'abstraction auquel il soumettait son objet n'était pas sans parenté avec la programmation.

À la différence des bâtiments et des automobiles, les logiciels ne sont pas des objets physiques, ils sont même parfaitement invisibles, comme l'a remarqué F. P. Brooks [24] ; les moyens qui ont été imaginés pour les dessiner n'ont jamais été satisfaisants. Dans les années 1960 ce fut l'ordinogramme, auquel succéda, avec la programmation structurée des années 1970, le diagramme arborescent, pour céder la place vingt ans plus tard au diagramme UML *(Unified Modeling Language)* inspiré de la programmation par objets des années 1990 : ces procédés seront évoqués plus longuement au chapitre 5, mais il est d'ores et déjà possible de dire qu'ils n'ont jamais pu prétendre au rôle joué par le dessin industriel pour l'industrie mécanique. Et, lorsqu'ils s'en rapprochaient, c'était au prix d'une complexité équivalente à celle de l'écriture du programme, ce qui dissuadait les programmeurs de s'en servir ; alors que l'avantage du dessin, c'est qu'il est beaucoup plus facile à réaliser que l'objet lui-même : moins d'énergie, moins de matière, moins de temps et moins d'argent.

Mais comme justement les logiciels ne sont pas des objets physiques, dont on pourrait montrer une maquette au donneur d'ordres, ils ont les avantages de l'abstraction.

Imaginons que l'on veuille construire un bâtiment sans très bien savoir dès le départ combien d'étages il aura. On va commencer par creuser des fondations pour une cabane de jardin, construire la cabane, puis imaginer de lui ajouter un étage, et ainsi de suite jusqu'à cinq ou six étages : le lecteur sent bien que ce n'est pas la bonne démarche, parce qu'ajouter de la profondeur aux fondations n'est pas facile à première vue, c'est même sans doute impossible, et il en va de même si l'on veut renforcer les murs porteurs. Bref, la législation du permis de construire est là pour interdire une telle démarche, et dans les pays où l'absence d'une telle législation n'est pas compensée par le respect de règles ancestrales issues de l'expérience, et où l'on construit ainsi, les accidents provoqués par les écroulements d'immeubles sont monnaie courante.

Eh bien cette démarche si inappropriée pour un bâtiment convient très bien pour un logiciel. Je me risquerai même à dire que c'est la meilleure, et que tous les logiciels vraiment bons ont été réalisés ainsi.

Pour résumer, par rapport au bâtiment ou à l'automobile, le logiciel possède une complexité de nature différente : il s'agit d'une complexité abstraite, plus difficile à maîtriser par l'imagination qui se trouve ici privée du secours que pourraient lui apporter dessins et maquettes. Décrire dans un document le détail de ce que sera un logiciel est une entreprise beaucoup plus difficile que pour un objet concret ; de plus le délai nécessaire pour la mener à bien est si long qu'entre-temps les desiderata du maître d'ouvrage auront changé.

2.4.3 Cycle de vie du logiciel

Une vision managériale périmée : le cycle en V

Mais si le logiciel offre une complexité plus difficile à appréhender que celle d'un objet matériel, il présente les avantages qui découlent de son caractère abstrait : il est infiniment malléable. Il est donc possible d'en réaliser une version préliminaire rudimentaire, livrée au maître d'ouvrage qui pourra l'essayer, découvrir d'ailleurs à cette occasion que ses besoins n'étaient pas ceux qu'il imaginait, et faire part de ses remarques et suggestions, qui donneront lieu à une nouvelle version plus élaborée, et ainsi de suite.

Procéder ainsi suppose évidemment que le maître d'ouvrage collabore très étroitement avec l'équipe de conception, et que l'activité de conception, comme le suggèrent les adeptes de l'*eXtreme Programming* [9], mais contrairement aux habitudes, englobe la programmation. Une telle démarche postule aussi un dispositif contractuel fondé sur la confiance mutuelle et un cahier des charges léger, ce qui n'est pas toujours possible, notamment dans un contexte de Marchés Publics, mais sans doute plus facile à obtenir dans le cas d'une réalisation par des équipes de la même société que le maître d'ouvrage.

Il est clair que cette démarche, que nous examinerons plus en détail au chapitre 5, est impossible si l'on adopte le cycle de développement « en V » hérité des métiers du bâtiment et préconisé par les méthodes traditionnelles. La figure 2.1Cycle de développement « en V » du logiciel selon la vieille écolefigure.2.1 reproduit le schéma traditionnel de ce cycle, qui se parcourt en commençant en haut à gauche par la spécification des choses à faire, et en terminant en haut à droite par la recette du projet. Entre-temps on aura procédé, par étapes aussi bien séparées que possible, à la conception globale, puis à la conception détaillée, enfin au « codage ».

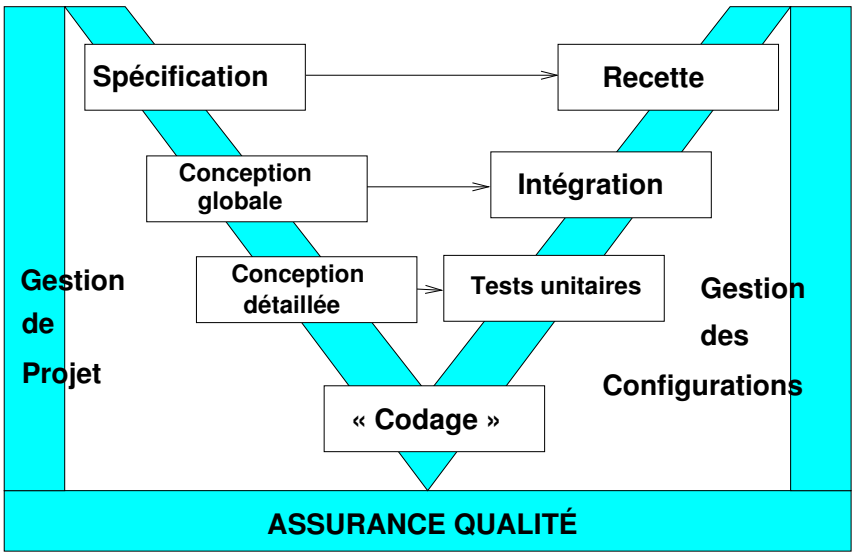

Figure 2.1 : Cycle de développement « en V » du logiciel selon la vieille école.

Ces différentes étapes seront confiées, au moins pour un projet important, à des personnes différentes, de rang hiérarchique décroissant au fur et à mesure que l'on descend le long de la branche de gauche du V. On remarque que l'activité de programmation, dévaluée par l'emploi du terme péjoratif « codage », qui suggère une transcription mécanique sans intelligence d'un document pourvu de « contenu », est, tout en bas, confiée à des tâcherons. Chaque rectangle de la branche de gauche, à l'exception du « codage », a sa contrepartie sur la branche de droite : le respect par le « codage » des prescriptions issues de la conception

détaillée est validé par les tests unitaires, l'intégration vérifie que les « livrables », validés par les tests unitaires, s'assemblent bien selon le plan prévu par la conception globale, enfin les hauts responsables prononcent la recette en faisant vérifier que l'ensemble correspond aux spécifications émises initialement. Typiquement, la maîtrise d'ouvrage rédigera les documents de spécification et de recette (les deux rectangles qui occupent les sommets des branches du V), la maîtrise d'œuvre sera chargée des tâches indiquées dans les rectangles inférieurs.

On observe que ce schéma, inspiré d'ouvrages destinés à des responsables de projets informatiques, ne fait apparaître nulle part de rôle dévolu à l'architecte.

Tout l'ensemble de l'édifice repose sur le socle de l'assurance qualité, qui définit les procédures d'acceptation des livrables aux différents étages de la branche de droite. Les colonnes de la gestion de projet et de la gestion des configurations sont sans doute là pour éviter le basculement de l'édifice. L'assurance qualité et la gestion de projet incombent tant à la maîtrise d'ouvrage qu'à la maîtrise d'œuvre, cependant que la gestion des configurations est du seul ressort de la seconde.

Cette démarche semble empreinte de bon sens, et elle l'est d'ailleurs, à condition qu'elle soit adoptée avec modération. Il est bien sûr préférable de définir l'objet à réaliser avant de commencer à le fabriquer et, si plusieurs équipes d'entreprises différentes doivent participer au projet, il est souhaitable que les missions des uns et des autres soient délimitées avec autant de précision que possible. S'il s'agit d'un objet matériel, bâtiment ou machine, le respect de ces règles est même indispensable, parce qu'il sera difficile de changer d'avis quant au plan d'un bâtiment au moment d'attaquer le deuxième étage.

Mais s'en tenir de façon rigide à ce schéma pour un projet informatique présente trois inconvénients et risque de conduire son adepte à quatre erreurs.

Les trois inconvénients du cycle de développement en V

La liste qui suit ne saurait être exhaustive, mais les trois faiblesses de la démarche traditionnelles énumérées ci-dessous nous paraissent emblématiques.

1. S'obliger à terminer chaque étape de la branche de gauche du V avant d'entamer la suivante, et s'interdire d'y revenir, c'est se priver de l'avantage principal du logiciel par rapport au bâtiment : on peut modifier le plan et les fondations au moment d'attaquer le deuxième étage ! Il ne faut pas abuser de cette latitude, elle a bien sûr un coût, mais il n'y a aucune raison de s'interdire d'en profiter.
2. Un projet informatique, même de taille modeste, peut difficilement être mené à bien en moins d'un an. Le temps moyen de réalisation d'un progiciel commercial est de l'ordre de sept ans. Le système d'information, auquel il est destiné, n'est pas de la même nature qu'un bâtiment ou qu'une automobile. Lorsqu'un bâtiment est terminé, il restera tel qu'il est pendant des dizaines ou des centaines d'années. Lorsqu'un modèle de voiture entre en production sur une chaîne construite à cet effet, il ne subira pas de modification importante pendant quelques années. Le système d'information est, et doit être, plus malléable. Un logiciel de comptabilité ou de gestion de paie doit s'adapter aux variations de la réglementation ou à l'apparition de nouvelles technologies d'accès aux données telles que le WWW. Si les équipes de la maîtrise d'œuvre ont travaillé imperturbablement pendant trois ans sur la base des spécifications initiales, il est à peu près sûr que le logiciel livré à l'issue de la recette sera déjà périmé.
3. Non seulement le logiciel livré sera inadapté, mais la spécification qui l'aura engendré aura coûté trop cher. Puisque le cahier des charges est immuable, il aura été élaboré avec le plus grand soin. Dans le contexte des marchés publics français, par exemple, lorsque le maître d'ouvrage constate avec dépit lors de la recette que ce

qui est livré est conforme aux spécifications, et doit donc être payé au prestataire qui livre, et que pourtant c'est inutilisable parce qu'entre-temps le contexte a changé, il rédige un avenant au marché pour adapter le logiciel aux nouvelles circonstances. Il se fait alors tancer par les bureaucrates chargés du contrôle, qui lui disent qu'« il aurait dû prévoir ». Prévoir quoi ? Un changement de majorité parlementaire qui a entraîné des modifications de la fiscalité ou du droit social ? Cette exigence est bien sûr stupide, mais pour la satisfaire le maître d'ouvrage va désormais élaborer des cahiers des charges d'un niveau de détail exagéré, qui spécifieront des logiciels excessivement paramétrables, donc lourds, complexes et trop chers. Tout cela est d'ailleurs vain, puisque la vie se chargera de toute façon de créer des cas de figure imprévus.

Les quatre erreurs dans la conduite de projet

Pas plus que pour les trois inconvénients, cette énumération des quatre erreurs n'épuise le sujet, toutefois elle met l'accent sur les raisons les plus importantes qui militent en faveur d'un renouvellement des méthodes.

1. Comment le maître d'ouvrage, angoissé à juste titre par l'idée d'avoir à rédiger des spécifications pour un système destiné à être mis en place plusieurs années plus tard, va-t-il essayer de rédiger un cahier des charges à l'épreuve de tous les événements prévisibles ou imprévisibles ? C'est là qu'il tombe le plus souvent dans l'ornière que lui conseillent d'ailleurs tous les bons auteurs : l'« analyse de l'existant ». Cette activité en apparence innocente rassure les chefs de projet angoissés devant les incertitudes de l'avenir et les subtilités de technologies qu'ils ne connaissent souvent pas suffisamment. Or l'existant c'est l'organisation issue du passé que l'on veut justement remplacer. Il serait idiot de l'ignorer totalement, mais lui consacrer trop de temps peut avoir une influence néfaste. Pire : la rédaction des spécifications est souvent confiée à des personnes issues du terrain, ancrées dans les pratiques du passé sur lesquelles elles n'ont pas un regard suffisamment critique. Dans ce cas, et il est courant, le cahier des charges va consister en l'énumération de ces procédures, qu'il faudrait justement réformer. La démarche du projet, loin de déboucher sur une rénovation du système d'information, va durcir et ossifier des pratiques qui n'étaient jusque là que de mauvaises habitudes et dont l'énoncé va désormais être gravé dans le marbre du cahier des charges. Le projet aura débouché sur un renforcement de la bureaucratie et des dispositifs paralysants.
2. Une autre mauvaise pratique prend généralement son essor à peu près au même moment dans la chronologie du projet : le « recensement (ou l'expression) des besoins ». Le lecteur pourrait croire à un paradoxe : il est évident que l'on ne construit pas un système d'information juste pour le plaisir, et qu'il est préférable d'avoir une idée de l'objectif poursuivi. Mais trop souvent la pratique qui se donne cours sous ce prétexte ne sert en rien à éclairer la cible visée, mais plutôt à recueillir les opinions et desiderata hétéroclites de personnes plus ou moins reliées à l'activité concernée, et à embrouiller la marche à suivre dans un galimatias de motivations privées ou collectives plus ou moins incohérentes. De toute façon, lorsque viendra le jour de la recette, ces motivations auront complètement changé et les personnes consultées (si elles n'ont pas de toute façon changé d'affectation dans l'entreprise) ne seront pas satisfaites. Bien sûr ce tableau peut varier selon le contexte : dans une entreprise de taille moyenne dirigée par son propriétaire, la personne habilitée à énoncer le besoin est bien identifiée et, surtout, il y a de bonnes chances pour qu'elle agisse de façon conséquente par rapport aux décisions qu'elle aura prises, puisque c'est son argent qui sera engagé – et s'il n'en est pas ainsi, la conséquence sera en quelque sorte morale, parce que celui qui aura commis l'erreur en sera la victime. À l'autre bout du spectre, dans une administration publique en proie à la concertation compulsive,

savoir ce que l'on veut vraiment faire est à peu près impossible, et il faut reconnaître que certaines commissions de fonctionnaires risquent d'être prêtes à ne pas reculer devant des décisions incohérentes, d'autant plus que leurs membres n'encourent aucun risque personnel pour les inconvénients qui pourraient en découler. Il est d'ailleurs reconnu depuis longtemps par les spécialistes des organisations que la prise de décision collégiale, loin d'engendrer la modération des décisions, suscite la montée aux extrêmes par un emballement rhétorique que ne vient contrebalancer aucune angoisse devant des responsabilités futures qui seront de toute façon assumées par d'autres ; les deux attitudes en jeu dans ce phénomène sont l'illusion d'unanimité ou au contraire la polarisation conflictuelle, on en trouvera une analyse bien documentée sous la plume de Christian Morel [84].

La démarche d'expression des besoins peut être fautive de façon encore plus patente : considérons un établissement de recherche de taille moyenne, composé d'une direction générale qui regroupe quelques centaines de personnes et des unités de recherche qui regroupent quelques milliers de chercheurs. Les départements de la direction générale, pour accomplir leur mission d'administration de la recherche, désirent, et c'est peut-être légitime, se doter d'un système d'information pour savoir ce que font tous ces chercheurs. Si on les interroge pour qu'ils expriment leurs besoins, nul doute que l'on obtienne une longue liste d'informations, sûrement pertinentes d'ailleurs. Mais qui va devoir fournir ces informations ? Les chercheurs, déjà submergés de tâches administratives dont l'utilité leur semble obscure, et qui risquent de mal accueillir ces nouveaux formulaires à remplir à propos de tout et du reste. L'expression des besoins sera ici franchement nuisible. Nous verrons ci-dessous à la page 109 l'exemple d'une enquête statistique, exercice analogue à celui que nous évoquons ici en cela qu'il y faut, comme ici, commencer par déterminer quelles sont les informations qui pourraient être accessibles sans trop de difficulté, puis convaincre leurs détenteurs de bien vouloir les donner – sans la prise en considération de cette nécessité nulle expression des besoins ne peut donner le moindre résultat.

3. Les deux erreurs ci-dessus, conjuguées entre elles, peuvent être utilement complétées par une attitude intellectuelle qui semble de prime abord assez saine, mais qui va se révéler comme une nouvelle erreur : le souci d'exhaustivité. Prenons comme exemple un projet de sécurité informatique destiné à évaluer et à améliorer la disponibilité d'un système d'information. Le responsable du projet pourra par exemple s'inspirer de la méthode EBIOS élaborée en France par la Direction centrale de la sécurité des systèmes d'information (DCSSI). Une telle méthode le conduira à dresser une liste des risques, à associer chacun de ces risques à des vulnérabilités, et à envisager les contre-mesures qu'il peut élaborer pour s'en prémunir, selon une formule pleine de bon sens et d'utilité :

$$\text{risque} = \frac{\text{menace} \times \text{vulnérabilité} \times \text{sensibilité}}{\text{contre-mesure}}$$

Cette conceptualisation est intéressante, mais elle peut engendrer la tentation de dresser une liste de risques ou une liste de vulnérabilités que l'on placera dans la colonne de gauche d'un tableau, afin d'en remplir la colonne de droite avec les contre-mesures appropriées.

Pourquoi cette démarche est-elle maladroite ? Parce que les risques et les menaces sont nombreux et souvent inconnus, alors que le répertoire des contre-mesures possibles est beaucoup plus réduit ; souvent, cela peut se résumer à cinq ou six grands thèmes : plan de sauvegarde des données, amélioration du stockage, aménagement d'un site de secours avec duplication des données à distance, administration correcte des serveurs (fermeture des services inutiles, séparation des privilèges, application

des correctifs de sécurité, surveillance des journaux), sécurisation du réseau (pare-feu, authentification forte, fermeture des services inutiles), sécurisation physique des locaux. Il est donc plus simple et plus efficace de partir de la table inverse de la précédente : mettre les contre-mesures dans la colonne de gauche, et énumérer dans la colonne de droite les risques éliminés par elles, ce qui évitera de payer un consultant pendant des mois pour élaborer la liste des centaines de risques plus ou moins réels que l'on peut envisager.

Cette démarche prétendue exhaustive qui consiste à examiner de longues listes d'items pour cocher des cases dans la colonne d'en face, c'est la façon de travailler des ordinateurs, rapides, systématiques et mécaniques ; l'intérêt du travail d'un professionnel compétent par rapport à l'ordinateur, c'est que l'homme réfléchit, a des idées et des connaissances, il est capable d'en tirer parti pour éliminer les cas non pertinents et regrouper les cas similaires de façon à leur appliquer le même traitement, bref il est capable d'avoir une vue synthétique des choses. Réduire le travail humain à des procédures de type informatique est tout à fait contre-productif.

4. Les trois erreurs ci-dessus peuvent se combiner avec d'autres pour engendrer une méta-erreur encore plus stérilisante : la rédaction d'un Schéma Directeur du Système d'Information. Dans les administrations publiques, commettre cette erreur est une obligation réglementaire. Que le lecteur ne se méprenne pas : mener une réflexion sur les objectifs à moyen terme qu'il semble raisonnable de fixer au Système d'Information ne peut être qu'une bonne pratique, et le faire régulièrement une saine discipline. Mais ce n'est pas de cette oreille que l'entendent les instances de contrôle de l'administration, pour qui l'édification d'un Système d'Information ne saurait être qu'une charge. D'ailleurs le système de comptabilité publique n'offre aucun moyen de prendre en considération la valeur du capital dont la puissance publique se sera dotée en contrepartie de cette dépense[3]. Dans ces conditions, le Schéma Directeur a surtout le rôle de digue destinée à fixer des limites durables aux projets informatiques, et par là à limiter la dépense. Le contribuable qui lit ces lignes pourra penser que limiter les dépenses des administrations n'est peut-être pas une mauvaise chose. Sans doute, mais dans un domaine où l'innovation technique est non seulement foisonnante, mais encore porteuse de réductions de coût soutenues, figer pour plusieurs années le périmètre des développements et, surtout, le catalogue des technologies mises à contribution pour les entreprendre, c'est se donner la certitude de payer trop cher, avec un surcoût de plus en plus important au fur et à mesure que passent les années. Nous avons là un nouvel exemple d'une pratique qui semble de bon sens, alors que sa mise en œuvre sans prise en compte des caractéristiques réelles des objets auxquels elle s'applique, par défaut de compétence technique, aboutit très souvent (et pas seulement dans l'administration) à un résultat opposé à celui qui était espéré.

   Évidemment, pour la bureaucratie tout ira très bien, les choses seront bien en ordre, même si elles ont coûté en fait beaucoup trop cher, mais comment le savoir, puisque l'on n'y connaît rien ?

2.4.4 La vraie nature de la conduite de projet

Comment détruire votre informatique

Au début de ma carrière, à la fin des années 1960, à l'Insee, j'ai eu l'occasion d'assister à une réorganisation de l'informatique par un prestigieux cabinet de conseil qu'à la suite d'Émile

3 J'ai longtemps cru que cette inaptitude à envisager la notion d'investissement était propre à l'administration française, mais la lecture des livres du prix Nobel d'Économie Joseph Stiglitz[107] m'a appris que l'administration fédérale américaine souffrait du même mal.

Landormy[66] j'affublerai du nom d'Alex Andréiev afin de rétablir envers les peuples slaves un équilibre compromis par les noms tous anglo-saxons des prestigieux cabinets. Avant la réorganisation, le Département Informatique fort de près de cent cinquante personnes était regroupé dans un bâtiment unique du sud de Paris (rue Boulitte). Les opérateurs y côtoyaient de vrais théoriciens de l'informatique, et de cette ambiance assez désordonnée émanaient une véritable effervescence intellectuelle et une émulation tant pour acquérir de nouveaux savoirs que pour réaliser de nouveaux programmes. Ce département était un des endroits de France les plus en pointe pour l'usage et les développements informatiques.

Par ailleurs, les autres départements de l'Insee trouvaient que l'informatique ne leur donnait pas satisfaction, mais cela, de toute façon, est vrai presque partout et presque toujours. Dans la plupart des institutions (ou divisions d'institution) vouées à la production d'information, le travail réel est effectué par l'informatique, et les départements situés en aval réagissent à leur impuissance par des récriminations contre cette puissance démiurgique. Il est donc pratiquement toujours possible, pour se débarrasser de son informatique, d'arguer des plaintes qu'elle suscite de la part des « utilisateurs », et peu de dirigeants se privent du recours à ce moyen de pouvoir.

Les réorganisateurs d'Alex Andréiev, requis par la Direction Générale de l'Insee (dont on rappelle qu'elle est une direction du Ministère de l'Économie et des Finances, au même titre que la Direction Générale des Impôts ou que la Direction de la Comptabilité Publique), n'eurent donc guère de difficulté à établir un constat de mauvais fonctionnement et à proposer un plan pour une organisation plus professionnelle. Le principe de cette réorganisation était, ô surprise, taylorien.

Chaque fonction du Département Informatique (développement de logiciels, applications, système d'exploitation et réseau, exploitation) fut découpée en deux tranches : une mince tranche « fonctionnelle » dévolue à la conception, et une tranche « opérationnelle » plus épaisse consacrée à la production. À la tranche fonctionnelle était aussi dévolue une mission de pilotage, terme goûté par certains managers parce qu'il flatte leurs fantasmes de toute-puissance.

Il me semble nécessaire de s'interroger ici sur la permanence du modèle taylorien de l'entreprise au sein des cabinets de conseil : en quoi sert-il leurs propres intérêts commerciaux ? quelle relation entretient-il avec leur propre modèle d'organisation, qui comporte une comptabilité minutieuse du temps de leurs collaborateurs ? Et la logique de leur démarche commerciale ne leur impose-t-elle pas de forcément préconiser à leur client une réorganisation ?

De retour du service militaire, et affecté à l'équipe chargée du système d'exploitation, je dus choisir entre une affectation fonctionnelle, c'est-à-dire avoir le droit de réfléchir mais pas de toucher aux ordinateurs, et une affectation opérationnelle où je pourrais toucher aux ordinateurs[4] mais en exécutant aveuglément des procédures décidées pas les fonctionnels. Pour être sûr que les choses soient séparées, elles étaient installées sur des sites différents, fonctionnels à la Direction Générale et opérationnels au centre de calcul. Cette organisation m'apprit un principe : si un travail est intéressant, il suffit de le diviser suffisamment pour obtenir plusieurs activités inintéressantes. Je réussis à échapper temporairement au dilemme en me faisant nommer fonctionnel en stage au centre de calcul au sein d'une équipe opérationnelle, mais ce n'était que reculer l'échéance fatale de la bureaucratisation.

Dans le domaine du développement logiciel, Alex Andréiev nous avait trouvés trop artisanaux : il était préconisé de monter des projets plus professionnels et de recourir à des prestations extérieures pour la réalisation des logiciels. Ce fut fait. Un cahier des charges fut rédigé pour la réalisation d'un logiciel de dépouillement d'enquêtes statistiques, un appel

4 Pour que cela soit intelligible au jeune lecteur, il faut rappeler qu'à cette époque il n'y avait ni micro-ordinateurs posés sur les bureaux, ni réseau, et qu'une installation de système se faisait de nuit, après réservation de la machine plusieurs semaines à l'avance.

d'offres lancé et une société de services choisie comme prestataire. Tout cela était extrêmement bien fait : cahier des charges rédigé par des gens hyper-compétents en informatique et en statistiques, consultation des départements utilisateurs, constitution d'une équipe de projet avec deux ingénieurs Insee de haute qualification et une dizaine d'ingénieurs du prestataire, eux aussi très qualifiés. Il convient de souligner que la plupart des projets que j'ai observés étaient loin de bénéficier de conditions initiales aussi favorables.

Tout fonctionna à merveille : au bout de trois ou quatre ans et de quelques millions de francs le logiciel fut livré, et comme le lecteur des lignes ci-dessus peut le prévoir, il déçut. Son mode d'emploi était fort lourd et nécessitait un apprentissage d'une difficulté comparable à celle d'un véritable langage de programmation, sans en procurer la souplesse et la puissance. Des logiciels d'analyse statistique plus agréables à utiliser et plus versatiles étaient apparus sur le marché. Ses seuls utilisateurs furent les exécutants auxquels le choix n'était pas laissé, à de rares exceptions près. Nous reviendrons sur les leçons à tirer du développement de ce logiciel à la page 110, mais disons ici que de cette expérience pourtant menée avec le plus grand sérieux et dans toutes les règles de l'art j'ai dû tirer trois conclusions : pour faire de l'informatique intéressante il fallait m'extraire des organisations alexo-andréievienne, si possible je devais devenir responsable de l'informatique pour pouvoir l'organiser à ma guise, et j'éviterai alors, dans toute la mesure du possible, de faire appel à des prestataires extérieurs.

La chance m'a été donnée de pouvoir réaliser ce programme pendant vingt ans, grâce à des directeurs d'établissements de recherche qui m'ont accordé leur confiance pour diriger leur informatique scientifique, et je crois pouvoir dire qu'ils s'en sont bien trouvés, et moi aussi. J'ai fini par être rattrapé et démasqué par les alexo-andréievienne, mais ceci est une autre histoire.

Vingt et quelques années plus tard, je me suis donc retrouvé dans un univers qui essayait de devenir conforme à la doctrine du prestigieux cabinet Alex Andréiev, et je constate que dans ce monde de l'informatique où soi-disant tout va de révolution en révolution, rien n'a changé si ce n'est le vocabulaire. Les projets de système d'information (locution inconnue dans les années 1970, notez-le) menés par des sociétés de service coûtent toujours aussi cher, les délais sont toujours autant dépassés, et le résultat final déçoit toujours autant les utilisateurs (c'est un euphémisme).

Les détails de la conduite de projet

Aujourd'hui (et depuis quelque temps déjà, mais cela va en s'aggravant) il n'est plus question que de *projet*. On travaille en *mode projet*, c'est-à-dire en appliquant les méthodes de conduite de projet, qui sont l'objet d'enseignements universitaires qui jettent pas mal de poudre aux yeux. Ne pas parler ce langage, c'est s'exposer à ne plus être considéré comme un professionnel. En quoi cela consiste-t-il ? En ceci :

1. Rédiger des documents pour préciser ce que l'on veut faire, à chaque étape, ainsi :
   - note d'argumentation, pour convaincre de l'opportunité de lancer le projet ;
   - note de cadrage, pour délimiter le périmètre fonctionnel affecté par le projet (c'est, au passage, un exemple de *parler projet*) ;
   - dossier de faisabilité.

   Une fois que la décision de lancer le projet a été prise :
   - cahier des charges ;
   - plan qualité du projet ;
   - planning du projet ;
   - spécifications générales et détaillées ;
   - plan de recette et plan de formation ;
   - différents dossiers techniques ;
   - documents de recette du projet ;
   - documents de déploiement (j'abrège).

2. Planifier : il existe des logiciels (et même des logiciels libres) de gestion de projet, qui permettent d'enregistrer les différentes étapes du projet et leurs durées, et d'en déduire un calendrier, avec le calcul des effectifs mobilisés pour chaque tranche de temps, etc. Ces logiciels peuvent bien sûr produire des diagrammes de Gantt ou de Pert.
3. Travail fusionnel : tous les participants au projet sont regroupés dans un espace de travail ouvert (« *open space* »), développent à grands cris, communiquent en temps réel et sont en « 100% réactivité » (oui, ils parlent ainsi).

La liste des documents énumérés au point 1 semble tout à fait raisonnable. Pour ce qui est des documents destinés à scander l'avancement du projet, nul ne pourra nier qu'il est bon de rendre explicite ce que l'on veut faire plutôt que de se lancer au jugé dans un projet mal évalué, et pour ce faire les documents cités ici (tirés d'un exemple réel un peu simplifié) semblent raisonnables.

Pour le point 2 je serai plutôt réservé : à la différence des adeptes de la conduite de projets, je pense que la réalisation d'un logiciel, fût-il destiné à prendre place dans un système d'information, est une activité de conception, en tant que telle rétive par nature à la planification ; il faut bien sûr tenter de juguler cette rétivité, mais un excès de planification, ou une planification trop précise ne peuvent que répandre des illusions et des déceptions et finalement stériliser le projet. J'exposerai un peu plus loin une hypothèse sur le rôle réel de ces diagrammes de planification.

Là où les choses peuvent se gâter, c'est lorsque la conduite de projet selon la méthode standard engendre l'apparition d'une profession particulière, le *chef de projet*, dont bien souvent la compétence va se limiter à la chefferie de projet, et qui va être chargé d'encadrer les *ressources*, pour exhiber le terme insupportable qui sert à ces gens-là pour parler des êtres humains qui travaillent sous leur responsabilité. Que l'on m'entende bien : le chef de projet est *indispensable* à la réussite du projet, mais dès lors que la position qui lui est assignée est essentiellement administrative, et le cas n'est pas rare, la production de documents va cesser d'être le support d'une activité plus importante de nature technique, la réalisation d'une application informatique, et va sombrer dans une bureaucratie envahissante, c'est-à-dire devenir une fin en soi. À partir de ce moment, la compétence technique devient une gêne dans le projet, parce qu'elle menace le pouvoir du chef de projet (et de ses chefs à lui), assis sur une montagne de papier – remarquons que dans une telle organisation, avec d'un côté des bureaucrates et de l'autre des développeurs, toutes les erreurs sont forcément attribuées aux développeurs, ne serait-ce que parce qu'ils sont en bout de chaîne, les rédacteurs de cahiers des charges ne peuvent pas avoir tort ; un dispositif qui a pour conséquence une situation aussi aberrante ne peut guère se justifier. Il va donc devenir urgent d'éliminer la compétence interne pour lui substituer une compétence externe fournie par une entreprise extérieure, et de ce fait plus docile puisque mercenaire ; il n'y aura plus « en interne » que des rédacteurs de cahiers des charges et de plannings. Mais ceci a bien sûr aussi des inconvénients, comme tout mercenariat : perte de compétence de l'entreprise sur des activités (le système d'information) dont on ne cesse pourtant de proclamer qu'elles sont stratégiques et vitales, coûts incontrôlables, disparition de toute vision relative à l'évolution des techniques, enkystement dans des solutions périmées que l'on ne sait plus comment remplacer, sclérose généralisée. Ce tableau clinique n'est hélas que trop facile à observer tant il est répandu.

Pour ce qui est du point 3 de la liste ci-dessus (regroupement de tous les participants au projet dans un « *open space* »), il s'agit surtout d'un outil de productivité destiné à renforcer le contrôle sur le personnel, avec la contrepartie (pas nécessairement involontaire) de créer des conditions où tout réel travail de création est impossible. Ce qui résout *ipso facto* le dilemme du point 2 examiné ci-dessus : puisque l'on n'a plus affaire à un travail de conception mais à un travail de bureau répétitif ordinaire, il redevient planifiable ! Le remplissage de formulaires vides de sens est une activité dont le rendement est calculable avec une bonne

précision. Sans oublier la signification concrète de l'aspect « 100% réactivité » : on est à la merci du téléphone pour obéir sans délai à toute injonction des « fonctionnels », ce qui rend impossible tout travail réel.

Une autre critique à l'encontre des méthodes de conduite de projet est souvent formulée par les développeurs : la hiérarchie du projet est très avide de documents, de *reporting* comme on dit, l'établissement de ces documents consomme énormément de temps, mais il n'y a le plus souvent aucun retour d'information vers les opérationnels ; tout laisse croire que ces documents n'ont qu'une justification administrative, alors que s'ils étaient vraiment utiles à l'accomplissement des objectifs du projet ils devraient engendrer un flux d'information dans les deux sens. L'aspect unilatéral du *reporting* donne à penser qu'il s'agit simplement d'un dispositif de surveillance du personnel, un peu comme le chronométrage dans les usines tayloriennes.

Un chef d'entreprise informatique de haute technicité, qui avait auparavant exercé des responsabilités importantes dans le dispositif public de recherche scientifique, me confiait un jour son idée sur la raison d'être véritable de ces dispositifs de conduite de projet : aujourd'hui, la direction du personnel par des méthodes autoritaires, du type « vous allez faire ceci ou cela parce que je suis votre chef et que je vous dis de le faire », fonctionne de plus en plus mal, et plus du tout dans les activités de conception. Alors le calendrier prévisionnel vient prendre la relève de manière douce : « tu vois, sur le planning, jusqu'à telle date les cases qui correspondent à telle tâche, qui t'est attribuée et que tu as acceptée en réunion de projet tel jour, eh bien ces cases sont jaunes, mais après cette date elles sont rouges, eh bien cela veut dire que les livrables de la tâche devront avoir été livrés à ce moment-là ». Cela semble tout bête, mais ne marche pas trop mal.

Là où la méthode de gestion de projet se révèle très utile, c'est entre les mains d'une entreprise de prestation de services, à l'encontre de ses clients. En général les prestataires maîtrisent beaucoup mieux la méthode que les clients, parce que c'est leur métier et qu'ils la pratiquent à longueur d'année, ce qui n'est jamais le cas des clients. Un calendrier prévisionnel habilement concocté peut donc servir à justifier l'entassement sur le projet de nombreux personnels aussi lourdement facturés que faiblement compétents et utiles, tandis que tout retard de livraison pourra être facilement imputé au client qui n'aura pas remis dans les délais contractuels tel obscur document que tout le monde avait oublié. Les dérives potentielles de la sous-traitance en matière de gestion de projet informatique seront développées plus amplement ci-dessous page 72 et suivantes, consacrées respectivement à l'externalisation et aux avantages et inconvénients comparés des prestations de services au forfait ou en régie.

Voici les leçons que je tire de ces péripéties. Au risque de me répéter, la division de l'activité informatique en « conception » réalisée par des « fonctionnels » et « production » réalisée par des tâcherons est inappropriée. La combinaison de la gestion de projet, du cycle de développement en V et du recours à un prestataire extérieur engagé au forfait sur la base d'un cahier des charges immuable est une recette pratiquement infaillible pour conduire à l'échec. Ce qui est triste pour le contribuable français, c'est que la réglementation de la commande publique rend cette démarche à peu près obligatoire.

En conclusion, la gestion de projet nous apparaît essentiellement, sans que cela nous surprenne, et une fois débarrassée de tout le verbiage pseudo-scientifique qui l'accompagne, comme un mécanisme du pouvoir de l'entreprise sur ses employés, et éventuellement comme une arme concurrentielle dans les relations contractuelles entre les entreprises sous-traitantes et leurs clients.

Littérature de conduite de projets

Réaliser l'inventaire des manuels de conduite de projet disponibles sur le marché serait une entreprise dont l'ampleur excède de beaucoup les dimensions du présent ouvrage, tant l'intensité de la demande en recettes de management prêtes à appliquer est grande dans

ce domaine. Leur lecture exhale en outre une grande monotonie, même si un certain bon sens et un certain humour n'en sont pas absents. Un ouvrage typique de ce champ est *Le Chef de projet paresseux... mais gagnant* [37], chez *Microsoft Press* ; parsemé de dessins satiriques souvent drôles, c'est un assez bon livre, propre à retenir jusqu'à la fin l'attention de lecteurs peu enclins à l'effort intellectuel. Les méthodes de travail qu'il préconise sont générales, et l'identité de son éditeur pourrait suggérer qu'elles s'appliquent aux projets informatiques ou de systèmes d'information, mais en fait la plupart des exemples concrets qui les illustrent sont empruntés aux industries de fabrication ou au secteur du bâtiment et des travaux publics. Si l'on tente d'imaginer la transposition de ces exemples au domaine informatique, on voit bien pourquoi cela ne peut pas donner les résultats espérés : tous les critères d'appréciation de la démarche du projet et de sa réalisation reposent sur des faits matériels et des objets concrets et mesurables, et l'on ne peut espérer trouver des critères comparables en informatique qu'au prix d'artefacts réducteurs. Ainsi on peut mesurer l'avancement d'un projet de développement au nombre de lignes de programme écrites, mais cela ne dira rien de leur justesse, ni encore moins de leur adaptation à l'objectif poursuivi, ce qui est bien sûr beaucoup plus important.

Une autre impression qui se dégage de la lecture de ce livre, c'est l'importance des procédures bureaucratiques : si l'on additionne le temps consacré aux réunions et celui dévolu à l'écriture et, peut-être, à la lecture des multiples rapports, notes et comptes-rendus dont la rédaction est préconisée, on se demande quand et par qui le travail va être fait. En réalité, si l'on garde à l'esprit le fait que cette méthode s'applique à des industries de fabrication ou de construction, l'énigme se dissipe : ces secteurs sont soumis à une division du travail héritée des siècles passées, il y a des terrassiers qui piochent (fusse avec des tracto-pelles), et des ingénieurs qui font des mesures et rédigent des rapports. Mais, comme nous avons déjà commencé à le voir, et la suite de cet ouvrage approfondira cette conception, la réalisation de logiciels et de systèmes d'information n'obéit pas aux mêmes contraintes, et ce type de division du travail y est voué à l'échec.

Post-scriptum : mesure des projets

Dès lors que par les bienfaits de la gestion de projet une activité de conception délicate et difficile à planifier a été ramenée à une routine bureaucratique, rien n'interdit plus de lui appliquer des méthodes de mesure tayloriennes. Au passage, clarifions bien notre propos : transformer une activité coûteuse en activité de routine peu qualifiée et planifiable, cela s'appelle améliorer la productivité, et c'est généralement considéré plutôt comme un bien. Il pourrait en être de même dans le cas que nous envisageons ici : le seul inconvénient, c'est que ladite activité de routine ne produit ici rien, ou à peu près rien, en tout cas pas du tout ce qui en était attendu, et le gain de productivité espéré, et même mesuré par des métriques fallacieuses, se révèle en définitive négatif.

Jacques Printz, spécialiste du génie logiciel, a produit plusieurs ouvrages [93, 94] consacrés à la mesure de l'activité des programmeurs, qui sont de bons exemples de ce que l'on peut faire de mieux dans cette direction : je crains fort que ce qui peut être mesuré par ces méthodes parfois fort raffinées ne corresponde pas exactement à ce qu'il serait souhaitable de mieux comprendre.

Faut-il céder au désespoir ?

La conclusion provisoire à laquelle nous a menés ce chapitre semble désespérante : les méthodes disponibles pour définir le travail à effectuer dans le cadre d'un projet informatique sont inadaptées, les cadres conceptuel et juridique qui devraient nous aider dans cette entreprise se révèlent être en fait des obstacles. Devons-nous céder au désespoir et retourner

à la plume d'oie et au boulier ? Les tenants de la nouvelle barbarie élitiste et technophobe, issus des meilleures écoles de la République, ne cessent de nous le suggérer plus ou moins insidieusement, cependant que les adeptes de la technoscience obscurantiste persistent aveuglément dans ces démarches inadaptées qui n'enrichissent que certaines sociétés de service et cabinets d'audit et d'organisation.

Le chef d'entreprise informatique de haute technicité déjà évoqué plus haut me fait remarquer que les critiques formulées ici, et également dans les chapitres suivants, à l'encontre des méthodes courantes de conduite de projet sont surtout valables dans le contexte du secteur public français. Il est vrai que beaucoup des observations qui sont à l'origine de ces critiques ont été faites au sein d'organismes publics. Je suis entièrement d'accord avec lui pour attribuer le rôle principal dans certains égarements de l'économie française à l'organisation de l'État, et notamment aux marchés publics. Mais justement l'État occupe dans ce pays une place importante et il n'est pas isolé du non-État par une cloison étanche. Il a donc largement contaminé les autres secteurs de la société, et particulièrement les grandes sociétés privées, presque toutes dirigées par des fonctionnaires, et pour lesquelles la concurrence du marché capitaliste est très atténuée, notamment du fait de leurs liens étroits avec l'État – et cette intimité est scandaleuse. Pour ce qui est des sociétés de services informatiques, la part la plus lucrative de leur marché est le secteur public, rendu encore plus rentable par son déficit de compétences. Ces relations impures contribuent à créer une situation malsaine qui tire l'ensemble de l'économie vers le bas en assurant des rentes à des agents économiques parasitaires et en fermant l'accès au marché (qu'il soit du travail ou de l'entreprise) à des acteurs potentiellement plus dynamiques qui pourraient créer plus de richesses.

Y a-t-il une autre voie ? Oui, et même plusieurs. Nous allons en parler, et d'ailleurs elle ne sont pas franchement inédites, mais elle n'ont jamais été bien populaires parce qu'elle sont fatigantes : elle demandent l'acquisition de compétences, puis du travail (par opposition à l'imitation de travail évoquée à la page 22). Elles feront l'objet de la suite de ce livre, non sans que nous ayons consacré le chapitre suivant à la normalisation et à la démarche qualité, détour inévitable à qui veut comprendre les méandres de l'encadrement du travail dans le monde contemporain.

Chapitre 3 Normalisation et démarche qualité

Sommaire

3.1 Genèse de la démarche

Nous ne saurions clore le rapide panorama des méthodes de contrôle du travail dressé au chapitre précédent sans évoquer la *démarche qualité*, qui s'est répandue voici une décennie et qui répand aujourd'hui une certaine déception.

La démarche qualité, et l'idée de normalisation à laquelle elle est étroitement associée, sont nées dans le monde industriel au XX^{e} siècle. Certes, le premier empereur chinois, Qin Shihuangdi (259-210 avant notre ère), avait déjà normalisé l'écartement des roues des chars (et prévu la peine de mort pour les contrevenants)[1], et en 1732 Louis XV en fait autant pour le diamètre des canons, cependant que la réorganisation des fabrications pour la marine de Louis XIV en 1689 peut être considérée comme un plan qualité précurseur. Le système métrique (1795) fut aussi une avancée majeure. Mais l'Association française de normalisation (AFNOR) ne voit le jour qu'en 1926, et l'*International Standardization Organization* (ISO) en 1947. Les Américains avaient créé le *National Institute of Standards and Technology* (NIST) en 1901. La première norme d'assurance qualité sera émise par l'armée américaine en 1959, et cette démarche traversera l'Atlantique dans le sillage du prestige associé au programme spatial Apollo, pour aboutir en 1987 à la publication de la première version des normes ISO 9000 dévolues au domaine « du management et de l'assurance de la qualité ».

Écartons dès l'abord le risque d'une confusion : le *contrôle de qualité* est une méthodologie à base de calcul de probabilités et de statistiques destinée à vérifier la conformité aux spécifications d'une production le plus souvent industrielle et de masse. Au stade le plus élémentaire il consiste à prélever des échantillons de la production, à les vérifier et à contrôler que le taux de défauts n'excède pas les seuils d'admissibilité. Il n'entretient que de lointains rapports avec la démarche qualité dont il est question ici.

La démarche qualité consiste à instaurer dans une entreprise un ensemble de procédures qui permettront d'établir et de vérifier son aptitude à fournir des produits satisfaisants pour ses clients. L'idée initiale était « on écrit ce que l'on fait, on fait ce que l'on a écrit ». Cette exigence a été atténuée parce qu'elle engendrait une paperasse invraisemblable, mais elle peut

1 Qin Shihuangdi avait également ordonné la destruction de tous les livres existant dans l'empire afin qu'ils soient réécrits avec un contenu plus conforme à ses souhaits. Ce Fils du Ciel était décidément un précurseur de génie !

cependant donner une première intuition de la démarche, de concert avec l'idée associée de traçabilité : s'il survient un incident, une erreur ou une faute, il faut pouvoir en retrouver l'origine afin de les corriger.

Pour ce faire, le fonctionnement de l'entreprise sera analysé afin d'identifier les processus qu'elle met en œuvre, leurs *inputs* et leurs produits, ainsi que les interactions entre eux. Cette analyse en processus permettra de modéliser le fonctionnement de l'entreprise, et ainsi de décrire dans des documents les flux d'information et de matières qui la parcourent, et en fin de compte ce que chaque acteur d'un processus doit recevoir (comme information et comme matière) pour effectuer sa mission, et ce qu'il doit livrer au processus suivant à l'issue de celle-ci.

Nous ne pouvons que le constater une fois encore, nous venons de tomber sur des idées qui nous apparaissent sous le jour favorable du bon sens et de la rationalité : qui pourrait contredire le projet de construire un modèle opérationnel de l'organisation de l'entreprise afin de mieux la comprendre et de pouvoir vérifier que les projets formulés sont bien exécutés comme prévu ? Comment douter que la livraison aux clients de produits satisfaisants et conformes à leurs commandes, de préférence dans les délais fixés, ne soit une condition de la survie de l'entreprise ? Et toute méthode qui permet de s'approcher de ces buts n'est-elle pas à adopter ? Sans aucun doute : pour que le résultat soit à la mesure de l'attente, il faut encore que la méthode soit appliquée avec suffisamment de réalisme et de sobriété pour ne pas introduire plus de désordre qu'elle n'est censée en supprimer.

3.2 Pour une normalisation modérée

De même, la normalisation est un principe propre à emporter l'adhésion. La non-normalisation est une plaie qui coûte très cher à quiconque veut construire un système d'une certaine complexité. Pour citer un exemple minuscule, j'ai dû réécrire des dizaines de pages de programmes Fortran parce que leur auteur avait utilisé deux dispositifs exotiques du langage qui n'étaient plus disponibles sur le nouvel ordinateur où il fallait désormais l'exécuter : c'est alors que j'ai rêvé à Ada, langage bien structuré et strictement normalisé. À une échelle plus vaste, des technologies telles que les réseaux informatiques et leur utilisation pour partager des documents à distance (le WWW), l'enregistrement vidéographique ou l'impression laser, n'ont pu se déployer à grande échelle qu'avec l'apparition de méthodes et de procédés universels admis par tous, que ce soit par l'effet de normes officielles ou par un accord entre industriels.

Si la normalisation a engendré une grande désillusion, c'est bien celle induite par l'expérience des normes de réseau de l'ISO. J'ai eu l'occasion à la fin des années 1980 d'avoir sous ma responsabilité une équipe qui travaillait avec des systèmes ISO (réseau *X25*, messagerie X400), et un ingénieur qui travaillait avec les systèmes de l'Internet (réseau UUCP et TCP/IP, messagerie SMTP). Au bout d'un an j'ai bien dû constater que le coût du système ISO était supérieur d'un ordre de grandeur à celui du système Internet, et surtout qu'il ne permettait pas vraiment d'utiliser le réseau, ne serait-ce que pour le courrier électronique. Cette conclusion ne nécessite pas une argumentation très serrée, quinze ans plus tard la cause était entendue et tout le monde avait pu faire les mêmes constatations.

Quelle était l'origine de ce fiasco des normes de réseau de l'ISO ? Les moyens engagés pour les définir et les implémenter avaient été considérables, les principaux opérateurs de télécommunications et plusieurs des principaux industriels de l'informatique et des télécommunications avaient fait travailler des centaines d'ingénieurs sur ce projet, alors que l'Internet apparaissait à l'époque comme une entreprise artisanale et désordonnée, ce qu'il était effectivement. L'analyse d'un phénomène aussi complexe excède largement le cadre du présent ouvrage, disons simplement que la démarche de l'ISO apparaissait singulièrement dépourvue de pragmatisme, par opposition à celle des groupes informels

qui définissaient les protocoles de l'Internet. Cette absence de pragmatisme et d'esprit tourné vers la pratique ne semblait pas totalement indépendante du fait que les comités de normalisation sont constitués de gens dont c'est en fin de compte l'activité professionnelle à plein temps, qu'ils y soient délégués par des industriels qui ont des intérêts dans le domaine visé ou qu'ils soient des fonctionnaires de la normalisation. Il résulte de cette situation la constitution d'un milieu social de la normalisation, assez coupé de la pratique, dont l'objectif risque de devenir son autoconservation, en l'occurrence produire de plus en plus de normes, de plus en plus exhaustives, de plus en plus épaisses, et en fin de compte de plus en plus difficiles à appliquer. Il est vraiment dommage d'en arriver là, parce que mes participations épisodiques à des comités de normalisation informatique rattachés de façon plus ou moins directe à l'ISO m'y ont fait rencontrer des ingénieurs de haute valeur et de grande compétence, dont l'activité aura malheureusement été souvent stérile, en tout cas dans le domaine informatique. À l'inverse, les normes de l'Internet, les *Requests for Comments (RFC)* [58], sont des documents assez informels rédigés le plus souvent par des praticiens et dont l'adoption résulte en dernière analyse de l'acceptation par les développeurs et les utilisateurs. Aussi étrange que cela puisse paraître, cela donne d'excellents résultats, qu'il serait peut-être imprudent d'étendre à d'autres domaines, mais en tout cas cela donne à réfléchir : nous y reviendrons ci-dessous à la page 124. Le dispositif de production de normes de l'IEEE *(Institute of Electrical and Electronics Engineers)* semble aussi donner d'assez bons résultats, puisque qu'il encadre notamment avec succès toute l'infrastructure des réseaux locaux (Ethernet, WiFi, etc.).

Bref, on peut dire que la normalisation bien faite, restée proche des usages et sans bureaucratie excessive, est la meilleure des choses parce qu'elle seule permet le déploiement technique à grande échelle dont l'Internet offre le meilleur exemple, et que la normalisation en circuit fermé, coupée du contact avec les usages, risque de mener à l'échec, comme l'architecture de réseau de l'ISO en a administré la preuve, sans doute en partie parce que les grands opérateurs de télécommunications institutionnels de la fin de l'époque du monopole croyaient que leur pouvoir en ce domaine était absolu. L'insuffisance de normalisation ou une normalisation trop peu indépendante peuvent freiner l'expansion des technologies les plus prometteuses, et c'est le cas de Java, dont l'inventeur a voulu garder le contrôle exclusif au détriment des développeurs indépendants, dont beaucoup se sont découragés et se sont tournés vers des technologies moins perfectionnées mais plus disponibles et plus ouvertes.

Il ne faudrait pas inférer des lignes ci-dessus que leur auteur préconise d'abandonner le processus de normalisation au bon vouloir de groupes spontanés : la normalisation exige une persévérance et une continuité que seule une organisation solide peut procurer. Les organes de normalisation de l'Internet, sur lesquels nous reviendrons à la page 124, se sont constitués de façon moins formelle que ceux de l'ISO, mais ils n'en sont pas moins dotés d'une grande robustesse.

L'architecture réseau de l'ISO fut peut-être un échec ; il n'en reste pas moins qu'il s'agissait d'une magnifique cathédrale technologique, dont certaines chapelles ont d'ailleurs été incorporées à l'Internet, comme la norme d'annuaire électronique *X500* qui a donné naissance à la norme LDAP *(Lightweight Directory Access Protocol)*, universellement adoptée aujourd'hui. Le modèle en sept couches de l'architecture ISO a introduit durablement un surcroît de rigueur intellectuelle dans l'approche des réseaux, et des générations d'étudiants et d'ingénieurs en tirent encore profit.

3.3 Morne qualité

La démarche qualité, à l'instar de la normalisation, peut être une excellente chose tant qu'elle n'est pas poussée par les qualiticiens à des excès qui risquent d'en faire une véritable nuisance. L'existence de qualiticiens professionnels est d'ailleurs sans doute le premier de ces

regrettables excès, parce qu'une démarche qualité véritablement assimilée par une entreprise devrait sans doute être mise en œuvre par les différents spécialistes de chaque métier qui s'y exerce. Comme pour la normalisation, l'apparition d'une corporation de professionnels de la qualité, instituée en France par exemple dans l'AFAQ [3] *(Association Française de l'Assurance Qualité)*, n'est pas vraiment une bonne chose.

Il est malheureusement difficile de parler avec chaleur du jeu de normes ISO 9000. Ouvrir un de ses textes officiels [4, 5] est une expérience démoralisante. Si la version de l'an 2000 des normes est censée simplifier la démarche, elle ne l'a pas rendue franchement laconique.

Il y a une certaine cuistrerie à vouloir donner à n'importe quoi une allure scientifique. Depuis que l'humanité pense, les plus grands philosophes et les plus grands savants ont étudié le concept de causalité et se sont posé à son propos une foule de questions dont certaines sont encore en suspens : les spécialistes de la qualité balaient d'un revers de main ces atermoiements d'une autre époque et vous proposent un « diagramme causes-effet » dont vous n'avez plus qu'à remplir les cases et hop, vous maîtrisez vos processus. Depuis quelques siècles est apparue l'idée de progrès, et là aussi un débat toujours ouvert concerne les meilleurs esprits. Un manager moderne n'a pas de temps à perdre avec ces billevesées : l'échelle des cinq étapes du progrès est là pour scander votre marche vers la qualité ISO 9000. Il faut vraiment s'accrocher à la table pour éviter que le livre ne vole à travers la pièce. Mais beaucoup de faux concepts de cette sorte, empilés dans un style sentencieux et compassé, mélangés avec des fragments empruntés à la loi informatique et libertés, à d'autres normes ISO, à des manuels d'économie et de statistiques « sans formules » pour managers, finissent par constituer une masse significative de papier qui a toute l'apparence de la connaissance sérieuse. On y chercherait en vain une indication pratique qui pourrait aider à entreprendre quoi que ce soit de réel, par contre il y a tout ce qu'il faut pour créer des processus bureaucratiques générateurs de formulaires à remplir et de rapports à rédiger et à approuver. C'en est même le but avoué !

Les scientifiques ont compris depuis longtemps le risque de l'apparition de pseudo-sciences, engendré par le système d'évaluation par les pairs. Le plus difficile est d'amorcer le processus. Il faut une masse critique de quelques personnes pourvues de positions officielles dans les structures de la science reconnue, que l'on pourra chercher parmi les scientifiques de disciplines existantes, d'un certain ge, si possible un peu mégalomanes et frustrés d'avoir obtenu une reconnaissance certaine mais insuffisante à leurs yeux. Le mieux, si possible, est d'en trouver dans au moins deux ou trois pays différents. On peut alors commencer à organiser de petits congrès, à s'échanger des étudiants, à se citer mutuellement et à chercher des appuis tactiques (pardon, des collaborations interdisciplinaires) auprès de disciplines suffisamment éloignées pour éviter les confrontations gênantes. Il faut aussi lancer des publications, et, avec les imprimantes laser, les brevets de Rank Xerox tombés dans le domaine public et le WWW, c'est devenu beaucoup plus facile qu'il y a quelques années. Le succès n'est pas garanti, mais il y a des exemples grandioses : sciences de gestion, sciences de l'éducation, intelligence artificielle... Incidemment, le processus de création d'une vraie science nouvelle n'est sans doute pas très différent, et c'est bien ce qui donne à une entreprise de ce type, menée avec méthode et persévérance, quelque chance d'aboutir.

La démarche qualité ne cherche pas vraiment à se placer dans le champ scientifique, mais sous sa forme « pure et dure » l'idée n'est pas à écarter qu'elle soit un succès de cette espèce. Encore une fois, elle est née de bonnes idées, et dans un contexte qui la justifiait totalement. Le lancement d'un vol habité vers la Lune est un projet d'une complexité telle et pour lequel un échec est si grave que la mise en place d'une structure de supervision lourde est bien sûr indispensable. À une échelle moindre, j'ai eu l'occasion de voir le manuel d'utilisation d'un avion de ligne : il s'agissait d'un petit camion aménagé en bibliothèque, que l'on amenait sur le tarmac pour le feuilleter et arriver à fermer la soute à bagages. Aujourd'hui ce serait peut-être un jeu de CD-ROMs. Le volume de cette documentation est sûrement justifié.

En même temps, il s'agit d'une littérature assez systématique et répétitive, parce que la complexité d'un avion consiste beaucoup en répétitions de multiples éléments semblables.

La démarche ISO 9000 est peut-être raisonnable dans un environnement industriel, parce que justement on y effectue beaucoup d'opérations répétitives, ce qui veut dire que le ratio *nombre d'opérations / nombre de procédures* aura une valeur élevée. De surcroît l'erreur peut y être mortelle (nous verrons des cas de systèmes d'information où une erreur peut être mortelle). La construction d'un système d'information est exactement le contraire d'une succession d'opérations répétitives. Par définition, l'information est ce qui est imprévu. L'annonce d'un événement certain contient une quantité d'information égale à zéro, au sens de Shannon. Lorsqu'un ingénieur conçoit un appareil mécanique, il s'efforce d'avoir le plus grand nombre possible de pièces identiques, afin d'en simplifier la production. Lorsqu'un développeur conçoit un logiciel, s'il identifie des traitements identiques, il les supprime pour n'en avoir plus qu'un exemplaire, et lorsque le logiciel a été ainsi réduit par élagage des branches répétitives, il n'est plus que de la complexité à l'état pur. Appliquer ISO 9000 à une activité de développement informatique est d'une part coûteux, d'autre part assez stérile, car on ne saisit que des aspects très extérieurs de cette activité et de ses résultats, ce qui n'aide guère à identifier les défauts et à les corriger. On trouvera dans le livre passionnant d'Isabelle Boydens [21] une critique solide de l'application à un processus de conception logicielle de la norme ISO 9000 conçue pour la production industrielle.

À ce sujet de la complexité du logiciel, il est des vérités que l'on n'ose pas penser et encore moins proférer parce que le public, même et surtout le public cultivé, ne les jugerait pas vraisemblables. Le 8 juillet 2004, Richard Stallman, le porte-parole de la Free Software Foundation, a prononcé dans l'enceinte des Rencontres Mondiales du Logiciel Libre à Bordeaux une conférence intitulée « Les dangers des brevets logiciels ». Après avoir rappelé le lien entre la notion de brevet et celle d'idée originale, il a comparé sous ce rapport les logiciels avec quelques entités traditionnellement brevetables. Par exemple dans l'industrie pharmaceutique l'originalité d'une idée aboutit à la formule chimique d'une molécule, qu'il est indubitablement possible de protéger par un brevet. Un avion de ligne comporte des centaines de milliers de pièces qui, pour certaines d'entre elles, sont couvertes par des brevets, soit peut-être des centaines, voire des milliers de brevets. Mais qu'est-ce qu'un logiciel, sinon une suite d'idées ? Un logiciel de taille moyenne, disons de quelques centaines de milliers de lignes, comporte de l'ordre de cent mille idées[2], il est de ce point de vue beaucoup plus complexe qu'une molécule biologique ou qu'un avion, hormis le fait qu'un avion moderne comporte *aussi* des millions de lignes de logiciel. Les logiciels de grande taille, notamment les systèmes d'exploitation, sont les objets techniques les plus complexes du monde d'aujourd'hui.

Pour en revenir aux tentatives d'appliquer la démarche qualité à l'informatique, il peut être tentant de découvrir, dans d'autres volets du fonctionnement du système d'information, des activités plus industrielles. L'exploitation est une cible de choix. Mais dans une exploitation informatique moderne, ce qui est répétitif est automatisé, et ce qui ne l'est pas, et qui est vraiment décisif, est l'activité de conception et de paramétrage des automates, activité très similaire à du développement logiciel ; et même, le plus souvent, c'en est : le paramétrage des listes de contrôle d'accès d'un routeur ou l'écriture d'un script pour arrêter les machines lorsque l'onduleur détecte une coupure de courant, c'est de la programmation,

2 On verra à la section 5.3 du chapitre 5 un exemple où il a été possible littéralement de *compter* les idées d'un logiciel, et d'avoir ainsi une estimation de sa densité intellectuelle : le noyau de sécurité du logiciel de pilotage de la ligne 14 du métro parisien, sans pilote à bord, a été développé par la méthode B, qui consiste à élaborer la preuve de la correction du programme en même temps que son code, ce qui fournit une démonstration formelle de justesse. L'écriture des 100 000 lignes de code engendra 30 000 obligations de preuve, dont la plupart furent satisfaites automatiquement par le système ; il resta 2 500 démonstrations rétives à l'automatisation, à effectuer par les ingénieurs. Il est possible de dire que ce logiciel comporte 30 000 idées, dont 2 500 idées difficiles, pour 100 000 lignes de texte.

et pas de la variété la plus facile. Appliquer à cette activité des procédures administratives lourdes, telles que prévues par ISO 9000, ne va guère aider à l'améliorer. Cela fera passer au premier plan les interactions avec les interlocuteurs extérieurs, plus faciles à identifier et à comptabiliser, aux dépens des tâches de fond, qui font en réalité la qualité du service.

Si les procédures ISO 9000 ne peuvent pas appréhender la teneur intime du travail informatique, mais seulement son enveloppe extérieure, elles risquent de n'être plus qu'un banal outil de vérification du fait que les personnes affectées à ces tâches sont bien présentes sur leur lieu de travail le bon nombre d'heures, et que pendant ce temps elles se consacrent bien à la mission confiée par leur employeur. Il faut alors se poser deux questions : ISO 9000 est-il l'outil le mieux adapté à cette fonction ? Mettre en place ce type de contrôle est-il un facteur décisif d'amélioration des résultats du travail informatique ? La suite de cet ouvrage s'efforcera de fournir des éléments de réponse à la seconde question. La première ne nous semble pas mériter un long examen.

3.4 Pour une démarche qualité adaptée aux situations réelles

La naissance de la démarche qualité au sein d'un projet aussi important qu'Apollo lui a donné une impulsion initiale très forte, assortie d'un grand prestige. La naissance de cercles intéressés par cette démarche a constitué un apport intéressant à la batterie d'outils dont disposent les ingénieurs, toujours attachés à améliorer leurs méthodes et leurs processus, puisque c'est en fait l'essence de leur métier. La déviation de la démarche qualité vers la lourdeur bureaucratique est le fruit vénéneux de son succès même : elle est devenue une institution, avec ses professionnels à plein temps qui ont eu intérêt à la développer et à la déployer le plus largement possible. Sous couvert d'ISO 9000, des positions de pouvoir bien payées ont pu être confiées à des personnes dépourvues de toute compétence bien identifiée, ce qui revient à dire qu'il leur a suffi de savoir lire et d'utiliser habilement cette aptitude pour acquérir une position dominante par rapport à des ingénieurs qui ont cru, à leur grand tort, que leur compétence durement acquise avait une valeur décisive pour leur employeur. Quelques idées en provenance d'ISO 9000 ont séduit les instances administratives, toujours attirées par des procédures qui leur permettraient de contrôler le travail, si possible par des moyens externes qui évitent d'avoir à en comprendre la teneur intime.

Donner un tel pouvoir à des personnes dépourvues de compétences précises a un autre avantage : la direction est assurée de leur fidélité parce que leur position ne tient qu'à elle, alors que des agents compétents risquent d'être plus indépendants.

Encore une fois, entendons-nous bien : savoir lire, vraiment lire, comprendre et assimiler ce qu'on lit, est une compétence rare, il n'est pas aberrant de confier à ceux qui la possèdent des responsabilités spéciales. Quiconque doit diriger une entreprise ou un département d'entreprise est confronté au paradoxe d'avoir à encadrer des spécialistes dont il ne maîtrise pas le métier, ou du moins pas tous les aspects de leur métier. Cette situation paradoxale est inhérente à la division du travail, il faut s'en accommoder, et pour cela créer un système de traductions, de correspondances, en bref de signes, qui permette au dirigeant incompétent dans tel ou tel secteur de désigner et de diriger, pour ainsi dire de l'extérieur, les activités d'un collaborateur compétent dans ce secteur. Cette situation est délicate et exige beaucoup de doigté si l'on ne veut pas qu'elle devienne conflictuelle. La démarche qualité peut avoir à cet égard un rôle positif.

Décrire l'extérieur d'un processus sans analyser son fonctionnement intime ni la sémantique des objets qu'il affecte, c'est une démarche d'abstraction éventuellement utile, même si elle comporte un aspect d'encapsulation de la compétence dans de l'incompétence qui peut paraître désagréable et qui comporte sûrement des limites.

En fin de compte nous arrivons à la conclusion que la démarche ISO 9000 est un mécanisme du pouvoir de la direction de l'entreprise sur son personnel, un peu comme le système des chronométreurs dans l'usine taylorienne. Ce n'est ni une surprise renversante ni un scandale en soi : simplement, la distribution du pouvoir qu'elle organise n'est pas la mieux adaptée à l'activité d'édification du système d'information ; ISO 9000 n'apparaît pas comme un bon système d'incitation des informaticiens au travail et, contrairement à ce qui se passe à l'usine, la contrainte n'est pas un moyen applicable ici.

Nous devons citer un autre facteur de pénétration de la démarche ISO 9000 dans les entreprises : la pression des grands cabinets d'audit internationaux. On comprend que ces cabinets soient attirés par des procédures normalisées d'évaluation d'une activité à partir d'éléments externes, et si l'explosion en vol scandaleuse d'Arthur Andersen n'avait jeté un certain discrédit sur leur activité, on pourrait dire que c'est légitime. Une version légère d'ISO 9000, limitée aux grands processus dont le fonctionnement peut intéresser les actionnaires, sans entrer dans le détail de l'organisation interne de l'entreprise, serait peut-être une évolution souhaitable.

De grands groupes industriels et certaines administrations ont exigé de leurs fournisseurs la certification ISO 9000 : à partir de là l'épidémie n'était plus enrayable, et elle s'est répandue dans toutes sortes d'organisations qui n'en avaient pas besoin, pour toutes sortes de processus qui ne s'y prêtaient pas.

Que l'on m'entende bien : toutes les organisations ont quelque chose à gagner dans l'adoption d'*une* démarche qualité adaptée à leur situation réelle, mais rares sont celles qui pourraient avoir intérêt à adopter une démarche qualité aussi coûteuse que celle décrite par ISO 9000, tant celle-ci est contaminée par le discours technocratique et par l'imitation de travail décrite par Alexandre Zinoviev [134].

3.5 La qualité totale des données : une utopie positiviste

La démarche qualité a été appliquée non seulement aux développements de logiciels, mais aussi aux bases de données et à leur contenu. Isabelle Boydens [21] a exploré aussi ces tentatives, et les lignes qui suivent doivent beaucoup à ses analyses. Nous examinerons d'abord l'approche *Total Data Quality Management (TDQM)* du *Massachusetts Institute of Technology (MIT)*, puis la *Field theory of information*, qui constitue une critique radicale de la théorie TDQM, bien qu'elle l'ait précédée dans le temps.

3.5.1 *Total Data Quality Management (TDQM)*

Le programme TDQM lancé par le MIT en 1991 se propose d'élaborer une théorie et des méthodes pour améliorer la qualité des bases de données. L'article le plus significatif pour qui veut cerner l'ambition de ce programme a été publié en 1996 dans les CACM [128] par R.Y. Wang et Y. Wand sous le titre « *Anchoring Data Quality Dimensions in Ontological Foundations* ». Nous voyons ici apparaître un usage du mot *ontologie* et de ses dérivés, que nous retrouverons à la page 116 et qui nous semble, là comme ici, abusif.

Cet article professe un positivisme[3] proprement incroyable :

« Le monde est fait de choses qui sont dotées de propriétés... Les propriétés sont représentées par des attributs qui sont des caractéristiques affectées aux choses par des humains... Les valeurs des attributs à un instant donné constituent l'état de la chose.

La connaissance d'une chose est appréhendée en termes de ses états. Toutes les combinaisons de valeurs des attributs ne sont pas possibles. Il y a des lois qui limitent les états

3 Comme je l'ai déjà signalé, je n'ignore pas qu'il existe par ailleurs un positivisme philosophique respectable.

autorisés à l'espace des états valides… Pour qu'un système d'information soit une bonne représentation d'un système du monde réel, ses états valides doivent refléter les états valides du système du monde réel…

Il y a imperfection de données lorsque la vision du système du monde réel qui peut être inférée d'un système d'information destiné à le représenter n'est pas conforme à la vision qui peut en être obtenue par l'observation directe. »

Ce texte est intéressant parce qu'il résume assez bien et formule explicitement les présupposés généralement implicites des théoriciens des systèmes d'information ; il comporte une métaphysique implicite, dont Isabelle Boydens donne un énoncé explicite (p. 62) :

« Une telle approche repose implicitement sur trois postulats :

— Le monde est composé d'éléments discrets, univoques, clairement identifiables et perceptibles.
— Les combinaisons et la connaissance de ces éléments sont gouvernées par des lois.
— Il est possible d'établir une relation biunivoque entre le réel observable et sa représentation informatique en vertu de l'isomorphisme qui les relierait l'un à l'autre.

… L'approche de Wand et Wang est tautologique… Poser un isomorphisme entre une base de données et l'objet observable correspondant permet ensuite d'analyser la qualité d'une base de données en termes d'un différentiel entre l'objet et sa représentation. »

Isabelle Boydens poursuit sa critique ainsi : il existe effectivement des cas où il est possible de confronter le contenu d'une base de données au système réel qu'elle représente ; ainsi on peut vérifier l'exactitude du catalogue d'une bibliothèque par une exploration exhaustive de ses rayons, mais c'est une opération coûteuse qu'il est hors de question de réitérer à chaque consultation. Et dans la majorité des cas une telle vérification est parfaitement impossible : Isabelle Boydens cite l'exemple de la comptabilité nationale[4] qui produit des grandeurs construites et estimées selon des méthodes approchées qui varient d'un pays à l'autre en fonction des sources disponibles et qui sont affectées conventionnellement à des intervalles de temps donnés, ou encore de la base des données collectées lors d'un tremblement de terre, qu'il est bien sûr impossible de reproduire. I. Boydens conclut que « la question de la correction de l'information empirique conduit inévitablement à une impasse », et nous nous rallierons à cette conclusion, non sans qu'elle soit encore renforcée par la section suivante. Incidemment, nous examinerons à la page 116 des points de vue voisins de celui de la théorie TDQM, et auxquels la réfutation d'Isabelle Boydens s'appliquera également.

3.5.2 *Field theory of information*

Après le premier choc pétrolier (1973-1974) l'opinion et les autorités américaines ont été prises de doute à l'égard du système statistique public, qui affirmait la présence d'importants stocks de produits pétroliers alors que l'automobiliste faisait l'expérience de la pénurie. Sous la présidence de Jimmy Carter, le *Department of Energy* entreprit une évaluation des bases de données qui dura cinq ans et examina 400 systèmes d'information et 2200 bases de données. De cet immense travail d'audit, qui eut recours à des méthodes statistiques créées pour les besoins d'analyse de ce volume énorme (pour l'époque) de données, émergea une démarche scientifique : la *Field theory of information* [77].

La conclusion principale de cet audit, qui fonde la théorie, est la mise au jour du processus par lequel une donnée devient de l'information (oserons-nous dire l'algorithme donnée/information ?). Une donnée apparemment valide, décrite par un vocabulaire stable sur lequel tout le monde semble d'accord, ne deviendra une information qu'après avoir subi

4 Il ne faut pas confondre *comptabilité nationale* et *comptabilité publique* : la seconde expression désigne le système comptable de la puissance publique, au sens de la comptabilité des entreprises, cependant que la comptabilité nationale est une branche de la science économique qui vise à donner de l'économie d'un pays une image globale au moyen d'agrégats comme le produit intérieur brut (PIB), notion bien connue, ou la formation brute de capital fixe (FBCF) qui récapitule les investissements des acteurs économiques.

une *interprétation* par un être humain qui ainsi lui conférera un *sens*. Et cette interprétation dépendra fortement de celui qui la donne, de ce qu'il cherche dans ces données, de la date à laquelle il les consulte. Bien sûr, ceci est vrai aussi et encore plus pour celui qui *constitue* la base de données. Les données ne sont pas un élément tangible de la réalité, mais un artefact construit avec une *intention* déterminée ; cette intention conditionne les interprétations qui pourront être faites de ces données. L'information est encore moins un élément tangible de la réalité, elle n'existe que dans l'esprit de ceux qui interprètent les données ou les textes, elle n'a aucune réalité objective.

Munis de cette conclusion, nous ne serons dès lors pas le moins du monde surpris de l'existence, pour citer Isabelle Boydens (p. 95), « d'un fossé "large mais invisible" entre les besoins en information et l'information elle-même. L'opacité du fossé est due à l'absence de contrôle et de vue simultanée sur le phénomène représenté, le processus de production de l'information et le processus d'exploitation de celle-ci. » Ainsi, « les bases de données relatives aux *ventes d'énergie* étaient confondues et couplées avec d'autres bases de données relatives à la *consommation en énergie*, ou encore, des données collectées dans le contexte des *inventaires* étaient exploitées dans d'autres systèmes en vue de mesurer la *distribution*. »

Ces erreurs d'analyse commises sur la base d'une confusion entre données et informations d'une part, d'une croyance naturaliste dans le caractère réel des données d'autre part, avaient mené droit à une disjonction majeure entre les prévisions des statisticiens et la réalité. Nous retrouverons de telles démarches erronées au chapitre 5 lorsque nous examinerons les *méthodes de spécification et de conception* du logiciel. Ces erreurs sont au principe de l'*utopie du système d'information* que nous analyserons au chapitre 7.

Signalons que le livre de Michel Volle *Analyse de données* [120] rejoint sur de nombreux points les analyses de la *Field theory of information*.

Chapitre 4 Le travail informatique

Sommaire

4.1 De la nature de l'informatique

4.1.1 Premières croyances

Les premiers ordinateurs, qui entrèrent en fonction à l'extrême fin des années 1940 et durant les années 1950, étaient consacrés à des travaux militaires ou scientifiques puisque, à cette époque, on pensait qu'ils seraient utiles essentiellement aux calculs des physiciens. Le projet phare de ces années fut SAGE *(Semi-Automatic Ground Environment)*, système de traitement des données issues des radars de défense anti-aérienne, destiné à prémunir les États-Unis contre une attaque nucléaire soviétique, dont le coût final atteignit huit milliards de dollars de l'époque. D'autres projets militaires d'une ampleur comparable permirent la naissance et l'expansion d'une population de professionnels de la programmation des ordinateurs et, au bout du compte, l'apparition d'une véritable industrie du logiciel, bien décrite par Martin Campbell-Kelly [25]. Cet afflux de financements militaires joua un rôle de premier plan dans l'élan initial de l'industrie informatique américaine.

Il ne fait aucun doute que lors de ces premières réalisations informatiques la part du logiciel fut considérablement sous-estimée tant sur le plan financier que sur ceux de la durée

et de la complexité du travail à accomplir. On s'imaginait en être quitte avec quelques dizaines de lignes de langage machine écrites sur un coin de table pour faire marcher les ordinateurs qui, eux, exhibaient les signes extérieurs de leur prix élevé en pesant des dizaines de tonnes et en consommant des centaines de kilowatts-heure d'électricité. Aussi les dépassements de coûts et de délais furent-ils la règle et engendrèrent-ils frustrations et déceptions ; c'est d'ailleurs encore le cas aujourd'hui.

Les premiers programmeurs étaient des scientifiques recrutés pour leurs compétences en analyse numérique et en algèbre linéaire, puisqu'il s'agissait essentiellement, croyait-on, de résoudre des équations différentielles et d'inverser des matrices. Ces problèmes mathématiques furent d'ailleurs résolus sans soulever de difficultés insurmontables, même si l'application de l'informatique aux problèmes numériques reste un domaine de recherche actif, et le restera tant que les physiciens continueront à inventer de nouvelles équations pour modéliser de nouveaux problèmes. Ce qui est erroné n'est donc pas d'attribuer à ces problèmes de numériciens une place dans la science informatique, c'est d'imaginer cette place comme centrale et absolument décisive. Une conséquence éminemment visible de cette erreur se donne à voir en France (terre particulièrement affectée par elle) où le principal institut public de recherche en informatique a été animé, puis dirigé, de sa création en 1967 jusqu'à la fin de l'année 2003 par des numériciens purs qui avaient de l'informatique une vision très unilatéralement mathématique et numérique, pour ne pas dire très peu informatique.

4.1.2 Comment l'informatique diffère des mathématiques

J'aimerais à l'occasion de cette analyse attirer l'attention du lecteur sur une question qui est une source constante de malentendus au sujet de la programmation.

Il y a fort longtemps, à l'Insee, un de mes collègues informaticiens débattait avec un statisticien, issu d'une grande école scientifique de la République, d'un projet informatique que le premier aurait à réaliser pour le second. L'informaticien expliquait le délai nécessaire à ce projet par la longueur des développements, due notamment à la nécessité de procéder à des essais pour éliminer les erreurs qui se seraient glissées dans les programmes. Cette démarche scandalisait le statisticien : « Quand je résous une équation différentielle, » disait-il, « je ne fais pas d'erreur, pourquoi en commettez-vous dans vos programmes ? ». L'auteur pourrait simplement rétorquer, d'expérience personnelle, qu'il arrive que l'on commette des erreurs en résolvant des équations différentielles, mais en fait la question n'est pas là. La démarche du calcul mathématique, pour qui en a acquis une maîtrise suffisante pour tel ou tel type de problème, comporte dans son déroulement même un dispositif d'auto-vérification. L'enchaînement des égalités qui conduisent à la solution obéit à des règles formelles et précises telles que les erreurs doivent, en principe, sauter aux yeux. L'imagination intervient dans l'élaboration de l'expression qui va faire progresser le calcul, et c'est pour cela que la résolution des problèmes mathématiques ne peut pas être confiée à un automate, fût-il un ordinateur, mais une fois cette expression rédigée et écrite au tableau, sa capacité à vérifier la relation d'égalité avec l'expression précédente est, pour quelqu'un qui s'y connaît, immédiatement perceptible.

Les preuves de programme

L'écriture de programmes informatiques obéit à de tout autres principes. Il convient de préciser cette affirmation pour la préserver d'une réfutation facile : du fait de sa naissance dans un milieu fortement influencé par le calcul mathématique, l'informatique a connu, et connaît encore, une activité assez intense de théoriciens qui entendent développer ce que l'on appelle des méthodes de preuve formelle, destinées à conférer à l'écriture de programmes la même capacité auto-vérificatrice que les équations mathématiques sont censées

posséder. La recherche dans le domaine de la démonstration automatique de théorèmes n'a jamais cessé d'être active. Il serait erroné de nier l'influence notable que ces recherches ont exercée sur l'orientation prise, notamment, par la conception des langages et des outils de programmation. Mais le programme que chacun utilise quotidiennement pour écrire son courrier ou son livre, naviguer sur le WWW, faire la comptabilité de son entreprise ou jouer sur sa console de jeux a selon toute vraisemblance été écrit par des programmeurs qui n'en avaient même jamais entendu parler. Ce serait peut-être moins vrai pour le logiciel qui pilote Ariane V ou une centrale nucléaire, sans d'ailleurs que cela garantisse le moins du monde l'absence d'erreurs de programmation, l'expérience le prouve. En fait, de tels logiciels comportent un si grand nombre de lignes de texte (un programme, c'est un texte) qu'il serait irréaliste d'en entreprendre la preuve, si tant est que ce soit possible ; on se contente d'essayer de « prouver » les parties jugées les plus critiques, quitte à s'apercevoir plus tard, lorsqu'une erreur survient et amène la destruction d'Ariane V et de sa charge utile, que la criticité n'était pas là où on l'attendait... et l'erreur pas dans le logiciel [67]. Bref, le logiciel, par nature, comporte des erreurs, encore et toujours. Nous aurons néanmoins l'occasion d'évoquer des méthodes réalistes de spécification formelle de programmes sûrs, par exemple à la section 5.3 p. 84 consacrée à la méthode B, et au chapitre 8 p. 121 qui abordera les projets d'informatique industrielle et technique.

Si les auteurs de programmes réels ignorent les méthodes de preuves formelles, ce n'est pas toujours par incompétence ou négligence : ces méthodes sont souvent inadaptées à leur travail, nombre d'entre elles ont été imaginées par des théoriciens universitaires très éloignés de leur pratique. Il ne fait aucun doute qu'un rapprochement entre ces deux démarches serait fructueux pour les uns et pour les autres, mais aujourd'hui en France cela ne peut arriver qu'exceptionnellement pour deux raisons : l'informatique universitaire y est encore souvent dominée (au moins idéologiquement) par des mathématiciens réformés surtout désireux de ne jamais avoir affaire à un ordinateur, d'autre part l'accès aux formations professionnelles informatiques est artificiellement limité par des effectifs insuffisants et par des exigences excessives de niveau en mathématiques, ce qui a pour conséquence qu'une grande proportion des professionnels de l'informatique ne possède pas de diplôme et n'a qu'une formation acquise sur le tas ou auprès des fournisseurs, c'est-à-dire totalement dépourvue de bases théoriques et du recul nécessaire à une vision d'ensemble.

Cette situation évolue depuis quelques années : la méthode B a commencé à obtenir des résultats convaincants. L'équipe LogiCal, qui associe des chercheurs de l'INRIA (Institut National de Recherche en Informatique et en Automatique) et du Laboratoire de Recherche en Informatique d'Orsay (LRI), a développé le logiciel *Coq*, un « assistant de preuve » prometteur doublé d'un langage de programmation sûr.

4.2 Programmation dans le monde réel

4.2.1 La vraie nature de la programmation des ordinateurs

Alors, comment s'écrivent les programmes informatiques ? Et d'ailleurs, qu'est-ce qu'une erreur de programmation ? Ces questions sont liées et elles sont, bien sûr, au cœur de notre préoccupation.

Si les énoncés des mathématiques sont auto-vérifiables, ou en tout cas nettement plus vérifiables que les énoncés du langage humain courant, c'est par la vertu de la méthode axiomatique, qui place au fondement de chaque théorie mathématique un corpus d'axiomes que les nouvelles propositions de la théorie devront respecter, et d'un formalisme rigoureux garantissant la solidité d'une démarche qui progresse d'hypothèses en conclusions par des transitions claires et sans ambiguïtés. Enfin, en principe. Kurt Gödel et d'autres ont montré les limites de cette méthode, qui n'en prête pas moins aux assertions des mathématiciens des fondations d'une exceptionnelle solidité, dans les limites de la théorie, s'entend.

Pour expliciter la démarche mathématique, les logiciens du XX^e^ siècle ont élaboré la notion de *système formel*, déjà suggérée par Leibniz plus de deux siècles auparavant. C'est important pour notre propos, parce que si les systèmes formels ont eu un succès modéré auprès des mathématiciens qui affectent le plus souvent de négliger cette question des fondements, ils sont à l'origine de la théorie des ordinateurs. Comme le dit l'encyclopédie *Hachette Multimédia* en ligne sur *Yahoo* [**hachette-yahoo**] :

« Le système formel est une notion logique fondamentale issue du double effort de constituer une logique mathématique et d'axiomatiser les théories mathématiques usuelles. Cet effort procède d'un idéal de rigueur – éviter les recours non contrôlés à l'intuition, expliciter toutes les hypothèses effectivement utilisées dans un raisonnement, par exemple – qui s'est diversement affirmé à partir de la fin du XIX^e^ siècle. Les travaux de Frege, de Dedekind, de Peano, de Hilbert, de Whitehead et Russell, de Gödel jalonnent brillamment cet effort. Ils contribuent à mettre au centre de l'intérêt des logiciens l'analyse de l'idée de démonstration, ou, comme on dit encore, de déduction, de dérivation ou d'inférence. »

Ou, avec plus de concision, Jean Kott nous dit que les systèmes formels sont constitués de « règles en nombre fini opérant sur des ensembles dénombrables » [64].

Nous pourrions ajouter à cette liste de logiciens l'Argentin Alonzo Church, créateur du λ-calcul, formalisme destiné à l'étude des fonctions mathématiques et qui a engendré le langage de programmation Lisp. Un disciple britannique de Church, Alan Turing, a cherché à préciser la nature des *procédures effectives* de Gödel, qui disait qu'un système était décidable s'il existait une procédure effective pour distinguer les propositions démontrables des autres. Inventer une procédure effective (un algorithme) consiste à déterminer un enchaînement d'opérations élémentaires qui exécuteront les calculs nécessaires à la solution d'un problème pour lequel existe une solution calculable (il y a des problèmes sans solution et des solutions incalculables). Une procédure effective doit recevoir une définition finie, et chacune de ses étapes doit pouvoir être exécutée mécaniquement. Pour clarifier cela, Turing a défini en 1936 le modèle théorique nommé *machine de Turing*, laquelle forme depuis lors avec le λ-calcul la base de la théorie des langages de programmation. Enfin le Hongrois naturalisé américain John von Neumann a élaboré en 1945 à partir de ces travaux théoriques le modèle concret de l'ordinateur, encore valable aujourd'hui et connu sous le nom d'*architecture de von Neumann*.

La science de la programmation des ordinateurs, qui constitue le cœur de l'informatique, est donc déterminée par un lien de filiation avec les mathématiques, ou plutôt (la nuance a son poids) avec l'étude logique des opérations des mathématiques. Le champ de cette étude logique qui a engendré l'informatique est organisé autour de la notion de *procédure effective*, dont le propos est de cerner précisément la suite des opérations concrètes par lesquelles s'effectue un calcul[1] ; ou, pour déjà parler informatique, la suite des opérations qui à partir de certaines *données* produiront certains *résultats*, constituant ainsi un *traitement*.

C'est pour définir sans ambiguïté des procédures effectives que Turing a imaginé la machine de Turing. Dès lors que l'on possédait le modèle théorique du calcul, la tentation de construire un automate capable de l'incarner était inévitable, et c'est ce que fit von Neumann.

Le propos d'un programme informatique n'est pas de démontrer un théorème à partir de certains axiomes et de certaines hypothèses, mais d'*effectuer un traitement* qui à partir de certaines *données* produira certains *résultats*. Pour se fixer les idées on pourra prendre comme exemple de traitement la multiplication de deux nombres selon la méthode apprise à l'école primaire, qui constitue un algorithme parfaitement transposable tel quel en programme informatique. Le traitement des données doit être *effectif*, c'est-à-dire que la méthode de

1 Dans les lignes qui précèdent le mot calcul était utilisé selon l'acception mathématique de ce terme. À partir d'ici nous l'emploierons au sens informatique plus large de *manipulation symbolique composée d'opérations de traduction et de réécriture*, bref, de traitement de données.

passage doit pouvoir aboutir pratiquement au résultat. Le traitement comportera des étapes, en nombre fini. Chaque étape consistera à élaborer à partir des données initiales un état du calcul ; les états successifs du calcul doivent conduire au résultat final ; pour progresser d'étape en étape ces états successifs doivent être enregistrés : pour l'écolier armé de son crayon, ce sera sur sa feuille de cahier ; pour l'ordinateur ce sera dans sa *mémoire*. L'organe de l'ordinateur qui effectue les calculs est nommé *processeur* ; chaque étape du traitement est accomplie par une action du processeur ; une action du processeur consiste à modifier, à *affecter* la mémoire, éventuellement après avoir consulté son état précédent, pour y consigner le nouvel état du calcul ; c'est très précisément ces opérations de consultation et d'affectation de la mémoire que formalise la machine de Turing ; le calcul sera terminé lorsque la mémoire sera dans un état qui contienne le résultat recherché. À certaines étapes du calcul la procédure effective doit comporter la possibilité de *choisir* entre deux actions possibles en fonction des résultats intermédiaires – par exemple y a-t-il une retenue ou pas : la machine de Turing est dotée de cette capacité. Ainsi on dit d'un automate capable de reproduire le fonctionnement d'une machine de Turing quelconque qu'il possède l'*équivalence turingienne*.

Nous avons introduit la notion de mémoire en notant qu'une action du processeur consistait à consulter ou à modifier l'état de la mémoire. Cette définition de l'action est très importante, elle trace la ligne de séparation entre la conception mathématique traditionnelle du calcul et la conception informatique liée à la notion de procédure effective. La mathématique ignore cette notion d'état, qui introduirait dans son univers d'abstraction un aspect physique totalement incongru.

L'informatique, et plus précisément la programmation des ordinateurs, confère au calcul une dimension concrète, effective. C'est un peu comme si le papier sur lequel le mathématicien inscrit les signes du calcul avec un crayon acquérait un statut théorique. Ce passage permanent de l'abstrait au concret est d'ailleurs l'agrément suprêmement fascinant de la programmation : dire c'est faire. J'énonce la formule d'une action, et la machine l'exécute.

Les lignes qui précèdent n'ont d'autre but que d'évoquer pour le lecteur ce qui apparente mathématiques et informatique, et ce qui les distingue. La dimension effective de la programmation des ordinateurs confère (impose) au calcul informatique une activité physique de construction de la solution, enregistrée dans les circuits électroniques de la mémoire de l'ordinateur (le mot anglais *storage*, emmagasinage, suggère mieux la nature de son rôle). La tâche du programmeur consiste donc à décrire les étapes successives de cette construction, au moyen d'un *langage de programmation*.

4.2.2 Langages de programmation

Un processeur quelconque est caractérisé par le jeu des actions élémentaires qu'il est capable d'effectuer. Ces actions élémentaires sont appelées les *primitives* du processeur, ou, si le processeur est une machine, les « instructions machine ». Un programme pour un processeur de von Neumann est une suite de primitives (d'instructions machine) du processeur destiné à l'exécuter. Chacune de ces instructions élémentaires correspond à un circuit logique du processeur considéré. L'ensemble des instructions machine et des règles de leur rédaction constitue le *langage machine*[2].

Une idée capitale du modèle de von Neumann, c'est d'enregistrer le texte du programme dans la mémoire de l'ordinateur, comme les données, et de considérer en fait ce texte exactement comme s'il s'agissait de données. Cette idée unificatrice permet d'appliquer aux programmes les mêmes traitements qu'aux données, de considérer le contenu d'un programme comme de l'information. Cela aura des conséquences évoquées plus loin.

2 Le lecteur désireux d'en savoir plus sur l'histoire et la morphologie technique des processeurs et des langages pourra se reporter par exemple à mon livre consacré aux ordinateurs et aux systèmes d'exploitation [14].

Les premiers programmes d'ordinateur étaient écrits en langage machine, mais cela avait plusieurs inconvénients. D'abord c'était extrêmement difficile et long, parce que les instructions machine sont très élémentaires et très proches de la structure interne du processeur, et induisent une formulation très rigide et malcommode. Ensuite, chaque processeur a un langage machine différent, et il faut donc réécrire le programme chaque fois que l'on change d'ordinateur.

Pour pallier ces difficultés furent inventés les langages évolués, dont le premier fut Fortran créé par John Backus en 1954, suivi de Lisp et Cobol, puis C, Ada, Java et bien d'autres. Un langage évolué est un méta-langage par rapport au langage machine, un programme en langage évolué doit être traduit en langage machine pour pouvoir être exécuté par un ordinateur. Le programme qui effectue cette traduction est généralement appelé compilateur. Pour exécuter un programme donné écrit en langage évolué sur plusieurs types d'ordinateurs, il suffit de disposer pour chaque type d'ordinateur d'un compilateur de ce langage, qui traduira le programme initial vers le bon langage machine. Un langage évolué est donc inséparable de ses compilateurs.

Il peut y avoir plusieurs niveaux de langages évolués, un langage de haut niveau étant traduit vers un langage de plus bas niveau, lequel sera par exemple traduit en langage machine. Tout cela est un peu plus compliqué dans la réalité, il faudrait parler du système d'exploitation qui s'interpose entre le programme et l'ordinateur, comme en fait un super-programme qui présente au programmeur une vision plus abstraite de l'ordinateur, et aussi d'un langage tel que Java qui est utilisé selon un schéma un peu différent, mais ces nuances ne sont pas ce qui importe ici, encore une fois je renvoie le lecteur qui en serait curieux à mon livre consacré à ce sujet, ou à d'autres ouvrages d'excellents auteurs, notamment ceux d'Andrew Tanenbaum [109, 111, 110].

4.2.3 Les erreurs de programmation

Les explications données ci-dessus nous permettent de préciser ce que sont les erreurs de programmation, et en quoi elles se distinguent des erreurs de calcul ou de logique qui peuvent affecter une démonstration mathématique.

Le texte d'un programme informatique est constitué d'instructions qui effectuent des actions, ou d'expressions dont l'évaluation produit des résultats, selon le langage utilisé. Ces actions (ou production de résultats) sont des modifications de l'état de la mémoire. Un langage pratiquement utilisable devra donc disposer d'un jeu de primitives suffisamment riche pour exprimer les différentes actions souhaitables, et cette richesse aura comme conséquence une grande complexité. L'initiation à un langage de programmation demande à un étudiant motivé plusieurs dizaines d'heures de cours et beaucoup plus encore d'exercices pratiques, pour en posséder juste les rudiments.

Le premier type d'erreur que le programmeur peut commettre est l'erreur de syntaxe, la violation des règles qui régissent le langage utilisé. Une erreur de ce type est bénigne, parce que le plus souvent le programme de traduction utilisé (le compilateur) la détectera et donnera des explications qui aideront à sa correction. On suivra pour l'éviter un conseil de Jacques Arsac : ne jamais programmer sans avoir le manuel du langage sous la main.

Un second type d'erreur est de nature sémantique : le programmeur a mal compris le manuel du langage, et il écrit un texte dont il pense qu'il va donner le résultat voulu, alors qu'il va donner un autre résultat, ou un programme faux, qui ne se termine pas, ou qui se termine par une erreur explicite. Une des qualités d'un programmeur expert, c'est de bien connaître la sémantique du langage qu'il utilise, connaissance qui demande des années de pratique assidue. Et il faut avoir le manuel du langage sous la main.

L'exécution d'un programme informatique peut aussi échouer pour une raison qui n'est pas à proprement parler une erreur de programmation, mais qui doit néanmoins retenir l'attention, parce que ce type d'incident est très fréquent : ce sont les échecs provoqués par

le contexte technique. Ainsi, un programme qui écrit des données sur disque peut échouer si le disque est plein. La saturation de la mémoire est une autre cause d'échec courante. En principe ces circonstances sont prévisibles par celui qui lance le programme, mais pas par celui qui l'écrit.

Mais une fois vaincues ces erreurs techniques, restent les vraies erreurs de programmation ; elles résultent d'une mauvaise interprétation de l'énoncé du problème à résoudre, ou du choix d'un algorithme inapproprié, ou d'une programmation fautive de l'algorithme, c'est-à-dire que le programmeur aura écrit un texte qui commande des actions qui ne conduisent pas au résultat voulu. On aura compris de ce qui précède que l'élaboration d'un programme pour résoudre un problème est une démarche de construction de la solution, par induction. Il faut imaginer les actions à effectuer ; les méthodes de preuve de programmes évoquées page 60 et sur lesquelles nous reviendrons à la section 5.3 page 84 ne pourront aider que dans certains cas assez particuliers, parce qu'elles sont trop lourdes pour être appliquées à l'ensemble d'un programme ; la plupart du temps il n'y a rien qui puisse fournir au programmeur l'équivalent du caractère auto-vérificateur du calcul mathématique, d'abord parce que la programmation n'est généralement pas une démarche déductive, mais aussi parce que le texte d'un programme est considérablement plus volumineux que celui d'un calcul mathématique ordinaire : la taille d'un petit programme vraiment utile pour un usage professionnel pourra difficilement descendre en dessous de 10 000 lignes de texte, un programme moyen aura plusieurs centaines de milliers de lignes, un système d'exploitation général plusieurs millions, plusieurs dizaines de millions si l'on y inclut les interfaces graphiques interactives.

Il y a par contre des méthodes de programmation destinées à limiter le risque d'erreur, qui feront l'objet de la section suivante. Le dernier mot reste sur ce sujet à Peter J. Denning : « il n y a pas de règles immuables à suivre pour écrire un programme logique, efficace et lisible. Il y a bien sûr des guides, et certains sont meilleurs que d'autres ; mais c'est le style du programmeur (ou son absence de style), sa clairvoyance (ou non), sa créativité (ou son absence) qui déterminent la qualité du produit final. »

4.2.4 Méthodes de programmation

L'invention et le perfectionnement des langages évolués allant de pair avec la prise de conscience de l'importance du logiciel et de sa difficulté, la question des langages de programmation, initialement posée en fonction du processeur, fut de plus en plus posée en fonction du programmeur. Pour écrire le plus vite possible des programmes comportant le moins d'erreurs possible, il faut disposer de langages adaptés. La linguistique de la programmation, c'est-à-dire la recherche dans le domaine de la conception, de la réalisation et de l'utilisation des langages de programmation, devint une des branches principales de la science informatique. La puissance limitée des premiers ordinateurs avait orienté cette recherche vers la conception de langages faciles à traduire, donc à compiler, ce qui facilitait la tâche de l'ordinateur. Dès les années 1960 la préoccupation principale devint la facilité à écrire des programmes aussi corrects et aussi lisibles que possible. L'évidence s'est en effet imposée qu'un programme est lu beaucoup plus souvent par des êtres humains, tels que son ou ses auteurs, ceux qui en assureront la maintenance ou qui devront le modifier, que par un compilateur. Les réalisations techniques marquantes de ce courant de pensée furent les langages Algol-60, Algol-68, PL/1, Pascal et Ada, les chercheurs les plus en vue, Edsger Dijkstra, C.A.R. Hoare, Peter Naur, Donald Knuth, Alan J. Perlis... La première qualité attendue d'un langage est, bien sûr, l'équivalence turingienne, c'est-à-dire l'aptitude à exprimer sans ambiguïtés tout algorithme formalisé par une machine de Turing, mais ces chercheurs ont ajouté à cette exigence l'expressivité et la lisibilité.

L'expressivité doit être entendue à la fois comme la puissance d'expression, la capacité à exprimer une opération complexe par un formalisme aussi simple que possible, et comme la

qualité de l'expression, le fait que le texte du formalisme suggère facilement sa signification au lecteur.

La linguistique de la programmation se préoccupe des aspects morphologiques et syntaxiques des langages, mais aussi des problèmes soulevés par la rédaction de textes assez longs, soit des méthodes de construction de programmes, dont il sera question à la section suivante de ce chapitre.

4.2.5 Méthodes de construction de programmes

Nous avons décrit ci-dessus le processus élémentaire de la programmation, celui qui consiste à écrire les instructions ou les expressions qui vont composer un programme. Un grand logiciel est constitué de centaines de milliers ou de millions de lignes de texte, le nombre de lignes étant souvent assez proche du nombre d'instructions ou d'expressions élémentaires, auquel il convient d'ajouter les lignes de commentaire qui sont du texte en langage humain, destiné à faciliter la lecture du programme.

Écrire un programme d'un seul tenant qui compterait des centaines de milliers de lignes serait très difficile, y retrouver et en corriger les erreurs serait une entreprise pratiquement impossible. Aussi a-t-on élaboré des méthodes pour découper de grands programmes en unités plus petites, plus faciles à écrire et à mettre au point séparément, puis réunies pour composer des ensembles plus importants, un peu comme un livre est découpé en parties, chapitres et sections, avec une table des matières et un index pour aider à s'y retrouver. Ces unités de programme peuvent être appelées sous-programmes.

Le premier problème à résoudre pour permettre le découpage des programmes en sous-programmes est d'organiser la circulation des données entre les sous-programmes selon des règles qui respectent le principe d'équivalence turingienne. Toujours dans l'idée de rechercher clarté, lisibilité et expressivité, des pistes nouvelles ont été explorées pour structurer l'information de façon à faciliter la tâche des humains, qu'ils en soient rédacteurs ou lecteurs.

La programmation structurée

Le premier courant de pensée qui associa la recherche d'une syntaxe claire et expressive à une organisation logique et commode des unités de programme fut la *programmation structurée* des années 1970, dont les leaders Dijkstra, Niklaus Wirth et Hoare étaient issus du courant Algol. Cette école fut représentée en France par Jacques Arsac et Bertrand Meyer.

Pour réaliser un programme, il semble logique d'adopter une démarche « de haut en bas » *(top-down)* : étant donné le problème à résoudre, je le divise en sous-problèmes plus élémentaires et faciles, que je divise eux-mêmes en sous-problèmes, jusqu'à ce que j'obtienne une collection de problèmes simples, dont je programme les solutions sous forme de sous-programmes qu'il ne me reste plus qu'à assembler. On reconnaîtra ici l'influence du *Discours de la Méthode* de René Descartes. Mais dans le cas de la construction d'un système informatique cette règle est plus facile à énoncer qu'à appliquer, parce qu'elle suppose connu le problème à résoudre, or c'est souvent là que gît la principale difficulté. Aussi la démarche inverse, de bas en haut *(bottom-up)*, est-elle utilisée également : je réalise des programmes qui modélisent un aspect partiel du problème, je vérifie que le résultat est satisfaisant, ce faisant je découvre sans doute des aspects du problème qui avaient échappé à la spécification. En fait, programmer un problème aide à le comprendre, et les approches de haut en bas et de bas en haut sont utiles toutes les deux.

En réalité la principale difficulté de la programmation réside dans la détermination de l'objet du programme à écrire. Paul Graham [53] l'exprime ainsi : « En programmation, comme dans beaucoup d'autres domaines, la difficulté n'est pas tant de résoudre les problèmes que de choisir les problèmes à résoudre ». Une approche fructueuse pourra consister

à programmer des fonctions générales, dont on sait qu'elles seront utiles de toute façon, sans se préoccuper de l'objectif précis à atteindre. Une fois ces fonctions disponibles, elles constitueront une bibliothèque de composants destinés à résoudre toute une série de problèmes de bas niveau (accès aux bases de données, affichage sur écran graphique, analyse des messages introduits par les utilisateurs), qui faciliteront grandement l'écriture de logiciels de plus haut niveau dont l'auteur aura l'esprit libéré de ces questions et pourra ainsi mieux penser au problème général. L'approche *bottom-up* revient en fait à construire d'abord, au moyen d'un langage de programmation général, un langage spécialisé doté d'opérations spécialement adaptées au problème traité, puis à traiter le problème dans ce langage *ad hoc* ; les langages de la famille Lisp – Scheme sont particulièrement bien adaptés à cette démarche, puisque l'on a pu écrire qu'il s'agissait de *langages de programmation programmables* [40].

La programmation structurée fut l'aboutissement des premiers efforts pour appliquer à l'activité de programmation une démarche intellectuelle systématique, en garantissant certains aspects de la cohérence formelle du programme et la permanence des propriétés de cohérence et d'équivalence turingienne lorsqu'un programme est divisé en sous-programmes. Les langages Pascal, Ada et Modula furent conçus selon les principes de la programmation structurée, avant de recevoir des extensions destinées à les adapter au modèle de la programmation par objets. Les principes de la programmation structurée peuvent être considérés comme un acquis de la science de construction de programmes.

La programmation par objets

Après la programmation structurée vint un autre courant significatif : la programmation par objets, inventée en Norvège à la fin des années 1960 par l'équipe de Simula (Ole-Johan Dahl et Kristen Nygaard), illustrée par les langages Smalltalk (Alan Kay à partir des années 1970), Eiffel (Bertrand Meyer, années 1980) et C++ (Bjarne Stroustrup) pour finalement connaître la consécration avec Java. Les langages de la famille Lisp et Scheme possèdent une approche objet qui leur est propre avec le modèle CLOS *(Common Lisp Object System)*. L'idée de la programmation par objets consiste à « encapsuler » un ensemble de données liées entre elles (par exemple, l'ensemble des données relatives à une même personne dans une base de données) avec les sous-programmes destinés à les traiter au sein d'une entité nommée *objet*[3]. Tous les objets relatifs à des personnes, pour poursuivre avec cet exemple, appartiennent à la *classe* `Personne`, qui constitue une sorte de modèle de ces objets analogues, qui sont nommés des *instances* de la classe `Personne`. Les sous-programmes associés aux objets sont appelés des *méthodes*, et s'ils sont les mêmes pour tous les objets de la classe ils figureront une seule fois pour toute la classe. La notion de classe est une extension de la notion de type[4] que l'on trouve en programmation structurée. Une nouvelle classe peut être obtenue par héritage d'une classe plus générale : ainsi, comme un élève est un cas particulier de personne, je peux construire la classe `Élève` à partir de la classe `Personne`, en héritant de tous les *attributs* de `Personne`, tels que date et lieu de naissance, adresse, etc. et en lui ajoutant des attributs spécifiques propres à la situation d'élève : classe, première langue, appartenance à l'association sportive ou au ciné-club, etc.

3 Le mot « objet » est un faux ami dangereux, puisque dans le langage courant il désigne un être du monde réel, alors que dans le langage informatique il désigne une représentation résultant d'une abstraction.

4 Le lecteur non spécialiste qui aurait quelques souvenirs de son programme de mathématiques de classe terminale peut voir le type comme une structure algébrique, telle que l'anneau des entiers relatifs ou le corps des réels. Un type est un nom donné à la collection des valeurs que peuvent prendre certaines entités, il est défini par les éléments suivants :

- Un ensemble de définition, appelé domaine du type. Les éléments du domaine sont les valeurs possibles du type (les nombres entiers, ou les chaînes de caractères...).
- Les opérations définies sur les éléments du type (l'addition, la multiplication et la comparaison pour des nombres, la concaténation et la comparaison pour des chaînes de caractères, par exemple).

Cette façon de structurer et de hiérarchiser l'information permet de mieux garantir la cohérence des données et des traitements, et ainsi d'engendrer automatiquement des sous-programmes standard qui tirent parti des règles de construction imposées aux objets. La hiérarchie des traitements n'est pas réalisée par des appels de sous-programmes, mais de la façon suivante : un objet envoie à un autre objet un message qui lui demande d'exécuter telle ou telle de ses méthodes.

Pour la programmation structurée comme pour la programmation par objets, le progrès dans la réalisation des programmes découle d'une meilleure organisation de l'information, d'une réduction du désordre (de l'entropie) dans les données et les sous-programmes. La première méthode s'applique principalement à l'organisation de l'information sur les traitements (le texte du programme), la seconde met l'accent sur l'organisation des données : en fait les deux approches sont complémentaires et la seconde a bénéficié des acquis de la première.

Pour apprécier les qualités et les défauts de tel ou tel langage de programmation, il convient d'avoir conscience des objectifs poursuivis par ses concepteurs, qui peuvent différer d'un langage à l'autre. Ainsi des langages comme Perl ou APL ont été conçus en visant la vitesse d'écriture maximum : ils sont concis et laconiques, pleins d'abréviations qui réduisent le nombre de caractères à écrire, mais cela donne un texte à peu près illisible qui rend très aléatoire la recherche d'une erreur dans un programme. À l'inverse, Ada a été conçu d'après un cahier des charges (émis par le *Department of Defense* américain, le DoD) qui mettait l'accent sur la réduction des coûts de maintenance du logiciel : écrire un programme Ada syntaxiquement correct est plus laborieux que de l'écrire en Perl, mais une fois que la syntaxe est correcte, les erreurs à l'exécution sont rares, et la structuration inhérente au langage facilite modifications et corrections ultérieures.

Excès dans la pensée

Il y a eu beaucoup de verbiage autour de l'aptitude supposée du modèle objet à « représenter naturellement le monde réel » : il faut beaucoup de naïveté pour croire à une telle assertion, à supposer même qu'elle ait un sens. Parmi ces fadaises, il a été beaucoup question d'un objet `véhicule`, construit par composition d'un objet `carrosserie`, d'un objet `moteur` et de quatre objets `roue`. Il était ensuite loisible de disserter sur le point de savoir si l'objet `tracteur` appartenait à la classe `Véhicule` ou à la classe `Matériel-Agricole`. En fait, l'objet informatique `tracteur` est une pure abstraction destinée à résumer pour les besoins de tel ou tel système d'information quelques données que nous pouvons éventuellement posséder sur un tracteur réel, piloté dans ses champs par un vrai paysan, ou en cours de fabrication dans une usine, ou objet d'une procédure d'immatriculation, ou d'un dossier de prêt bancaire. Cet objet `tracteur` sera selon nos décisions arbitraires, sans doute liées aux objectifs visés par le système d'information que nous construisons, une instance de la classe `Matériel-Agricole` ou de la classe `Véhicule`, ou encore de la classe `Immobilisation`, et le lien entre lui et le tracteur réel est une pure convention. Dans tout cela il n'y a que de l'information sur des choses, mais pas de choses du tout, et le grand avantage de l'information par rapport aux choses, justement, c'est qu'elle est abstraite, et que l'on peut en faire ce que l'on veut sans autre effort que le dire ou l'écrire, c'est-à-dire sans avoir à produire d'effet physique.

4.2.6 Où gît la difficulté de la programmation

La programmation et la construction de grands programmes ne sont pas des disciplines faciles. Et il serait vain d'espérer maîtriser la seconde sans avoir au préalable acquis la première. Les cursus de Génie logiciel, destinés à l'acquisition de la seconde de ces disciplines, voient affluer des étudiants qui n'ont jamais écrit une ligne de programme de leur vie, mais qui

voudraient bien, avec un diplôme au titre ronflant et un petit costume anthracite, devenir informaticiens en chef. Cette prétention est bien sûr ridicule, ce qui ne l'empêchera peut-être pas d'être satisfaite d'ailleurs.

Il existe un champ de recherche relatif à la programmation par l'utilisateur final ; il concerne les procédés qui permettent de rendre la programmation accessible dans une certaine mesure à des personnes dépourvues d'une véritable formation dans ce domaine. Les résultats obtenus par ces chercheurs indiquent que ce qui est difficile dans la programmation, ce n'est pas d'écrire une expression ou une petite séquence d'instructions qui vont réaliser une opération même assez complexe. La meilleure preuve en est la façon dont les utilisateurs les plus variés arrivent à se servir d'un tableur : même sans être programmeurs, beaucoup arrivent asez bien à programmer des formules assez complexes dans les cases d'un feuille de calcul. L'usage de logiciels statistiques ou de bases de données en montre d'autres exemples.

Là où les choses deviennent plus difficiles, c'est lorsqu'il s'agit de combiner plusieurs actions pour réaliser un programme plus vaste. Organiser un texte assez long qui doit décrire des situations où il y aura des choix à faire et des actions différentes selon les choix, répéter des actions pour en appliquer l'effet à des données variées en qualité et en quantité, c'est vraiment là que gît la difficulté – accompagnée en fait d'une autre difficulté tout aussi grande : la maîtrise des entrées et sorties d'un programme. Découper un programme en sous-programmes, organiser des données abondantes et variées d'une façon adaptée aux traitements qu'elles auront à subir, constituent des arts savants dont la maîtrise requiert un apprentissage théorique et une expérience pratique. Les méthodes à objets accroissent la puissance de ces procédés, mais pas forcément leur simplicité.

Un langage de programmation n'est pas seulement un moyen de demander à l'ordinateur le résultat d'un calcul : rédiger le texte d'un programme est aussi un moyen de noter et d'organiser ses idées, puis de les partager avec d'autres personnes. Pour permettre la rédaction harmonieuse de programmes éventuellement volumineux et complexes, un langage doit être pourvu des mécanismes suivants :

— des expressions primitives, qui représentent les objets élémentaires du langage ;
— des moyens de composition, aussi nommés constructions, qui permettent de construire des expressions composées à partir d'expressions plus simples ;
— des moyens d'abstraction pour nommer des objets simples ou composés, y faire référence dans d'autres parties du programme et effectuer sur eux des opérations logiques, mathématiques ou autres. Nous donnerons à ces opérations sur les objets du programme le nom de calcul, sans restreindre ce terme à sa stricte acception mathématique.

4.2.7 Rôle de l'abstraction

> *Abstraction : Action d'abstraire, opération intellectuelle par laquelle, dans un objet, on isole un caractère pour ne considérer que ce caractère ; résultat de cette action.* (É. Littré, [76])

Nous venons d'en parler, le processus d'abstraction, et le calcul sur les entités abstraites qui en résultent, sont au cœur de la problématique informatique. Programmer un problème du monde réel, qu'il s'agisse de mécanique quantique ou de comptabilité, consiste toujours à en identifier les éléments programmables pour ne plus considérer qu'eux, dont on construira un modèle abstrait, objet du programme. Il convient de s'interroger sur la nature et le statut de l'abstraction.

L'abstraction, dont j'emprunte à Émile Littré la définition ci-dessus, permet d'extraire d'un univers concret plus ou moins foisonnant des modèles d'objet auxquels la pensée pourra plus facilement s'appliquer, et par là elle est au cœur de la démarche scientifique.

Elle permet aussi de créer des classes ou des catégories d'objets. Ainsi au sein de la classe[5] des arbres je peux ne m'intéresser qu'à certaines caractéristiques, notamment celles du feuillage, et créer les sous-classes « conifères » et « feuillus ». Si mes définitions sont bien construites, je pourrai formuler des énoncés relatifs aux conifères en général, et ces énoncés pourront s'appliquer aux pins, aux sapins, aux épicéas, etc.

Chacun de nous fabrique des abstractions tous les jours, ne serait-ce que pour classer les personnes qu'il rencontre. Mais cette activité instinctive, naturelle, nous est masquée par notre éducation qui nous transmet de l'abstrait (les théories mises au point par des « génies ») et qui nous détourne de la pratique (explicite et lucide) de l'abstraction. D'autre part, le raisonnement abstrait systématique est une élaboration culturelle, transmise par l'éducation, et selon sa formation et ses aptitudes chaque individu possédera un entraînement plus ou moins grand à ce type de raisonnement. La programmation des ordinateurs exige un niveau d'accoutumance au raisonnement abstrait plutôt élevé, tel que peuvent en posséder des personnes ayant reçu une formation scientifique, et c'est ce qui fait que de telles personnes pourront éprouver moins de difficulté pour aborder cette activité, même si leur stock de connaissances ne leur confère pas un avantage décisif.

Les études scientifiques peuvent conférer à ceux qui en ont bénéficié une autre tournure d'esprit favorable : l'habitude de considérer le savoir et les connaissances comme des éléments de communication et d'échange, par opposition à la tradition artisanale encore très présente dans les métiers, pour laquelle les savoir-faire sont des secrets de fabrique. Edsger Dijkstra a rapproché dans un article célèbre cette situation de la création, à partir du XII^{e} siècle, des universités, qui dispensaient publiquement des savoirs qui auparavant étaient la propriété privée et secrète d'entrepreneurs de la science. Parmi les méthodes de construction de programmes que nous évoquerons, celles qui donnent les meilleurs résultats reposent notamment sur des échanges intenses de connaissances, et les développeurs qui ne seraient pas enclins à ceux-ci auraient du mal à s'adapter à celles-là.

Hormis cet entraînement très général au raisonnement abstrait, qui peut être acquis au cours d'études scientifiques, mais aussi tout autrement, par la pratique du dessin industriel ou de la linguistique, par exemple, rien dans les cursus secondaires français actuels ne prépare réellement à l'acquisition des bases de la programmation, et il n'y a donc aucune raison de restreindre l'accès des premiers cycles à orientation informatique aux titulaires de baccalauréats scientifiques.

Si la France avait voulu prendre au sérieux l'informatique et la formation des informaticiens, elle aurait suivi les recommandations formulées par Jacques Arsac il y a au moins vingt-cinq ans en créant un CAPES et une agrégation d'informatique (ce sont les diplômes qui permettent d'enseigner dans les établissements secondaires). Il faudrait apprendre à programmer tout comme on apprend à parler l'anglais : parce que c'est utile, parce que sans cela il y a beaucoup de choses que l'on ne peut ni comprendre, ni faire. Au lieu de cela on a préféré saupoudrer les cursus de quelques cours de sensibilisation clinquants et démagogiques.

4.3 Les illusions de la spécification

Les sections précédentes ont évoqué les inventions des langages évolués, de la programmation structurée et de la programmation par objets, qui toutes ont représenté des progrès dans l'art de la programmation en améliorant la productivité des programmeurs, la justesse des programmes et la facilité à les modifier et à les corriger en cas d'erreur.

Ces améliorations concernent l'activité des programmeurs : il n'en reste pas moins que la programmation, même facilitée par des langages bien conçus et des méthodes simplificatrices,

5 Le mot classe n'est pas employé ici dans son acception botanique.

reste une activité hautement complexe, qui demande de grandes capacités d'abstraction et beaucoup de temps pour un résultat finalement assez incertain.

4.3.1 Un modèle de division du travail

Nous avons écrit plus haut que la détermination de l'objet du programme à écrire était la principale difficulté de la programmation, et nous ajoutons ici que cette détermination doit être écrite dans le programme même, c'est d'ailleurs pour cela que c'est si difficile. Un aphorisme du folklore informatique en donne une idée : « Avec les ordinateurs, ce qui est ennuyeux, c'est qu'ils font ce qu'on leur demande, pas ce qu'on aurait voulu. »

Le coût économique de cette difficulté a logiquement engendré la recherche de moyens de la contourner. Que la difficulté intellectuelle réside dans la programmation, perçue comme l'étape de fabrication du produit final, était difficile à comprendre pour des managers habitués au processus industriel. Dans l'industrie mécanique, la difficulté intellectuelle réside au bureau d'études, où des ingénieurs hautement qualifiés conçoivent les objets à réaliser. Une fois la conception terminée, ils disposent d'un excellent outil pour transmettre les consignes de réalisation aux ouvriers : c'est le dessin industriel, ou le programme de machine à commande numérique qui lui a généralement succédé. Comme nous l'avons vu au chapitre précédent, nous pouvons dire que le système industriel taylorien organise la séparation stricte de la conception et de la fabrication. La conception effectuée par les ingénieurs se concrétise dans des dessins (ou aujourd'hui des programmes de machines à commande numérique) qui permettent aux régleurs des ateliers de paramétrer les machines de telle sorte que les opérateurs (c'est ainsi qu'aujourd'hui on nomme les ouvriers spécialisés) puissent fabriquer les pièces sans aucune marge de manœuvre. La transmission des consignes est sans ambiguïté aucune (du moins en théorie).

4.3.2 Essais de division du travail en informatique

Les managers qui ont cherché à organiser le travail informatique se sont tous plus ou moins inspirés de ce modèle industriel : ils ont cherché à inventer le procédé qui serait à la programmation ce que le dessin industriel ou la CAO sont à la fraiseuse, à l'étau-limeur ou à la machine à commande numérique.

Le degré zéro de l'informatique taylorienne

Dans les années 1960-1970 la division du travail était entre les programmeurs qui écrivaient des programmes d'un côté, et de l'autre les analystes qui produisaient des organigrammes, c'est-à-dire des schémas censés représenter l'enchaînement des opérations élémentaires du programme, que les programmeurs n'auraient plus qu'à « coder ». L'emploi dans ce contexte des termes coder, codage et codeur vise à dévaloriser l'acte de programmer, à suggérer qu'il n'est qu'une opération mécanique dont l'intelligence serait toute contenue dans l'organigramme produit par les analystes du service études, et que les programmeurs du service production n'auraient qu'une tâche mécanique de transposition à accomplir. D'ailleurs, pour joindre l'acte à la parole et donner à cette façon de voir une valeur autoréalisatrice, on s'empressa de recruter des programmeurs sous-qualifiés et de les entasser dans des ateliers taylorisés. Le résultat fut les millions de lignes de Cobol au style dit « en plat de spaghettis » qui encore aujourd'hui assurent une bonne partie de la gestion des banques et des compagnies d'assurance.

J'ai eu l'occasion de collaborer avec un collègue qui travaillait dans un tel contexte au début des années 1980, au sein d'un établissement financier. Il me cita l'exemple du programme qui créait le fichier quotidien des compensations inter-bancaires, destiné à permettre à l'établissement de se faire payer les chèques tirés sur d'autres banques qu'il avait reçus en paiement de ses clients. Le montant quotidien de l'opération se chiffrait en

millions de francs. Au moins une fois par semaine le programme échouait, ce qui obligeait à financer le retard de paiement au prix de l'argent au jour le jour. Autant dire que le coût du dysfonctionnement était colossal, équivalent au salaire de dizaines de programmeurs. Mais corriger l'erreur était proprement impossible, parce qu'elle était enfouie dans un programme monolithique énorme dont le texte source en Cobol était perdu depuis longtemps et dont l'organigramme disponible correspondait à une version périmée ; en outre le programmeur qui en était l'auteur et qui aurait peut-être pu intervenir avait quitté la société. Toute intervention aurait pu déclencher une catastrophe encore plus grave.

Cette anecdote, loin d'être exceptionnelle, reflétait la situation standard de l'informatique de gestion jusqu'aux années 1990, et il s'en faut de beaucoup que ce cas appartienne au passé.

Il faut ajouter que l'outil prétendu de conception, l'organigramme, était particulièrement indigent. Il consistait à aligner des rectangles et des losanges censés représenter les actions successives du programme et ses embranchements, c'est-à-dire qu'au mieux il était redondant par rapport au texte du programme, et la plupart du temps il était faux. Trop détaillé pour donner une vision d'ensemble, il ne l'était pas assez pour permettre une transcription automatique en texte de programme.

Première solution palliative : externaliser

L'ampleur des dégts provoqués par cette organisation du travail fut telle que chacun chercha des palliatifs. Les entreprises qui avaient tenté d'assurer elles-mêmes maîtrise d'ouvrage et maîtrise d'œuvre se trouvaient à la tête de logiciels inadaptés, difficiles à maintenir et bientôt périmés, ainsi que d'effectifs importants de personnels à la qualification et au potentiel d'évolution incertains. Les ingénieurs commerciaux des sociétés de prestations de service n'eurent guère de difficulté à les convaincre que ce n'était pas la méthode qui était mauvaise, mais le fait même d'entreprendre soi-même le travail d'informatisation, et qu'il serait bien plus judicieux de confier cette mission à des entreprises spécialisées.

La légende a longtemps couru, et court encore, en France, que la France posséderait une puissante industrie du logiciel. Cette illusion est produite par le chiffre d'affaires important des Sociétés de Services et d'Ingénierie Informatiques (SSII), interprété par des statisticiens et des économistes peu informés des réalités de l'informatique comme celui d'une illusoire industrie du logiciel. En réalité l'activité réelle de beaucoup de ces SSII est assez peu différente de celle des agences de personnel intérimaire, à tel point que certaines ont dû affronter les tribunaux pour éviter de justesse la qualification de leur activité en service d'intérim, ce qui aurait entraîné des conséquences lourdes et défavorables en matière de droit du travail et de fiscalité. Ces SSII, loin de créer de la plus-value en mettant des logiciels nouveaux sur le marché, vendent essentiellement du service. L'analyse de leur chiffre d'affaires, certes exceptionnellement élevé si on le compare aux valeurs observées dans les autres pays de l'OCDE, révèle une clientèle particulièrement abondante et assidue dans le secteur de l'administration publique, qui a accueilli avec un enthousiasme particulier l'idée de se débarrasser de toute compétence informatique. Cet objectif « incompétence totale » est d'ailleurs aujourd'hui pratiquement atteint, puisque dans les DSI la plupart des personnes formées à l'informatique deviennent des gestionnaires de contrats et ne maîtrisent plus du tout les techniques utilisées par les fournisseurs – lesquels sont d'autant plus à l'aise pour dicter leur loi ; si ces personnes dotées de compétences refusent le « projet professionnel » qui leur est ainsi proposé, elles sont repoussées vers la périphérie du dispositif.

Plusieurs facteurs concourent à cet enthousiasme dans l'éradication de la compétence interne des entreprises. Traditionnellement, les milieux dirigeants de l'administration française considèrent la compétence technique comme un attribut indigne d'eux, bon pour des tâcherons. En être dépourvu est une condition nécessaire pour accéder aux plus hautes responsabilités. Le lecteur pourra trouver des portraits pris sur le vif de tels dirigeants sur le

site de Michel Volle, comme par exemple ici [125, 118]. Comme les dirigeants des plus grandes entreprises françaises se recrutent dans la haute administration (issus de l'ENA : Conseil d'Etat, Inspection des Finances, Cour des Comptes etc. ; issus de l'X : Corps des Mines, des Ponts-et-Chaussées, etc.) cette culture de l'incompétence irrigue également celles-ci. Dans ce contexte, dire à des dirigeants qu'ils n'ont pas besoin d'être compétents, ni de posséder trop de compétence dans leur entreprise, en tout cas pas parmi leurs cadres dirigeants, ne peut que les conforter dans leur penchant traditionnel.

Il faut dire pour être équitable que les informaticiens ont aussi souvent abusé de leur pouvoir, qui devenait considérable, et qu'ils ont contribué eux-mêmes de façon importante à scier la branche sur laquelle ils étaient assis. Pour une entreprise, détenir des compétences n'est pas une fin en soi ; ceux qui les détiennent doivent les faire évoluer pour tenir compte des innovations techniques autant que des évolutions de l'entreprise ; la présence d'une population importante d'informaticiens qui ne voulaient pas ou ne pouvaient pas s'adapter aux nouvelles orientations a souvent été une charge insupportable pour leurs employeurs, et cette expérience n'a pas peu contribué à leur pulsion d'externalisation. Nous reviendrons sur cet aspect de la question informatique à la page 95.

À cette situation générale vient s'ajouter vis-à-vis de l'informatique ce qu'il faut bien appeler une haine particulière. Mon collègue Norbert Paquel, statisticien, économiste, spécialiste des systèmes d'information et observateur fin et exercé de l'économie française, m'a expliqué un jour que, pour que l'informatique marche bien dans ce pays, il aurait fallu que dès le début des années 1950 soit créée une école d'application de l'École Polytechnique dédiée à l'informatique, qui aurait alimenté un corps d'informaticiens de l'État, civils ou militaires. Ce corps aurait développé ses solidarités, ses implantations, ses terrains de chasse gardée et ses filières, avec ses voies traditionnelles de pantouflage, bref, l'informatique serait devenu un élément normal du système militaro-industriel français, à l'image de l'Armement, des Télécommunications ou des Ponts et Chaussées. Faute d'avoir suivi ce parcours, l'informatique n'a jamais atteint en France une respectabilité considérable, et, comme par ailleurs elle se développait et procurait aux marginaux qui choisissaient d'y faire carrière d'assez belles situations et des promotions rapides, ces succès perçus comme illégitimes engendraient envie, rancœur et animosité dans les rangs des corporations qui se considéraient elles-mêmes comme plus respectables et plus méritantes.

J'ai eu l'occasion d'observer maintes manifestations de cette détestation de l'informatique, sous des formes très diverses, une des plus curieuses étant l'annonce de la « fin de l'informatique », ou sa variante, la fin des salaires élevés pour les informaticiens. Dans la bouche de dirigeants, de telles prophéties peuvent prendre une valeur auto-réalisatrice, et effectivement, de façon récurrente, administrations et grandes entreprises, saisies par une mode panurgique, liquident leurs équipes d'informaticiens et externalisent à tout va, en proclamant haut et fort que c'est la voie de la modernité. Tout aussi régulièrement, à l'issue d'un délai que j'estime en moyenne à quatre ou cinq ans, les mêmes sont amenés à constater les résultats catastrophiques de cette politique, et, sans tambours ni trompettes, essaient de reconstituer ce qui peut l'être. Bien sûr, dans le climat social actuel, entre-temps les équipes dirigeantes ont été renouvelées au moins deux fois, ce qui dispense de toute réflexion auto-critique et de toute analyse approfondie.

Il existe une forme pseudo-savante de détestation de l'informatique : la proclamation hors contexte du *paradoxe de Solow*, du nom de son auteur l'économiste américain (Nobel 1987) Robert Solow, qui a dit en 1982 « *You can see the computer age everywhere but in the productivity statistics.* » (« on voit la révolution informatique partout, sauf dans les statistiques de productivité » ; la référence écrite est dans la *New York Review of Books*, 12 juillet 1987). Au début des années 1980 régnait l'informatique centralisée, assez mal maîtrisée par les entreprises ; en d'autres termes les innovations apportées par l'informatique n'avaient pas encore été assimilées, et de ce fait elles n'avaient pas encore porté leurs fruits. Depuis les choses ont malgré tout changé, et d'ailleurs Robert Solow a publiquement modifié son

diagnostic. Mais chez les cuistres ignorants, son ancien apophtegme voisine avec le détestable truisme sur les chercheurs qui cherchent et ceux qui trouvent. On consultera avec profit, à ce sujet, un article que Michel Volle a consacré à l'économie du système d'information [121], ainsi qu'un texte de Claude Rochet [97] assorti de nombreuses références accessibles en ligne.

Le débat sur le paradoxe de Solow s'est déplacé vers la question de l'inégale aptitude des économies américaine et européenne à tirer profit des innovations informatiques, mais l'opportunité d'y avoir recours n'est plus une question depuis longtemps : sans l'informatique, l'Airbus ne peut ni être conçu, ni être construit, ni voler, ni être vendu ; les banques devraient recruter des millions d'employés si elles voulaient se passer d'ordinateurs, ainsi d'ailleurs que les administrations fiscales qui ont pu grce à l'automatisation multiplier par plus de 10 le nombre de contribuables.

Pour en revenir à la question de l'utilité des compétences en informatique au sein de l'entreprise, il ne faudrait pas que le lecteur retire des lignes qui précèdent l'idée qu'il faille rejeter toute idée d'externalisation, bien au contraire. Il existe tout d'abord une forme d'externalisation qui est presque toujours hautement recommandable lorsqu'elle est possible et pratiquée raisonnablement, c'est le recours à des logiciels ou à des systèmes qui existent déjà, comme les progiciels : nous aurons l'occasion d'y revenir au chapitre 6. Il y a aussi des activités qu'une entreprise moderne aura tout intérêt à confier à des spécialistes extérieurs. Mais pour qu'une externalisation soit un succès, il faut avoir une idée la plus précise possible de ce que l'on veut externaliser. Il vaut mieux aussi que ce ne soit pas un élément trop stratégique du dispositif de l'entreprise, parce qu'il y aurait alors un risque de perdre le contrôle d'un élément clé. Il est souvent facile d'externaliser des choses dont la définition est assez simple, parce que très circonscrite : le nettoyage des locaux, la mise sous enveloppe et l'expédition d'envois en volume, l'entretien d'un parc automobile. Ce peut être aussi une intervention technique ponctuelle très spécialisée au sein d'une équipe qui ne possède pas cette compétence précise. Mais l'externalisation du système d'information, ou même simplement de ses composantes techniques, ne répond visiblement pas à ces critères. Les externalisations réussies concernent pour la plupart des activités qui ne demandent aucune compétence particulière ; dès lors qu'un travail révèle une complexité, l'intérêt pour une entreprise (publique ou privée) est de le capitaliser, donc de se l'approprier (*knowledge management*, veille, etc.)

Régie ou forfait : à bon chat, bon rat

Lorsqu'une entreprise confie un projet informatique à une société de services, situation déjà évoquée à la page 44, il y a deux types de solutions pour l'organisation du travail et de sa facturation :

— au forfait, le prestataire s'engage, sur la base d'un cahier des charges détaillé, à réaliser le système demandé dans un délai donné pour une somme fixée une fois pour toutes au départ, à lui de mettre en place les moyens appropriés pour mener le projet à bien ; il a une obligation de résultat ;

— en régie, le prestataire a plutôt une obligation de moyens, il met à la disposition du client un certain nombre d'ingénieurs et de programmeurs sous la conduite d'un chef de projet, et la facturation a lieu à la journée, en quelque sorte comme pour un taxi, et le travail est fini quand il est fini ; c'est en fait de la délégation de personnel plus ou moins déguisée.

Les contrats au forfait ou en régie sont à la mode alternativement ; l'administration française s'interdit les contrats en régie ; en fait l'un comme l'autre système peuvent réussir ou échouer.

Si le contrat est au forfait et que le client ne dispose pas des compétences et des effectifs suffisants pour participer activement au projet, le prestataire peut être tenté d'interpréter de

façon très restrictive le cahier des charges et ses propres obligations aux termes du contrat, et au contraire de manière extensive et pointilleuse les obligations du client, en termes de disponibilité des interlocuteurs, de fourniture de documents et de dates pour l'exécution de ces obligations. Si les équipes du client participent activement au travail et que l'estime réciproque règne entre les participants, ce type de dérive est moins probable et de toute façon plus limité, parce que chacun est en mesure d'apprécier le travail à faire et son volume. Mais, dans un contexte où chaque entreprise lutte pour sa survie et où, pour une société de services, avoir des employés en attente de travail (en « inter-contrat ») est une lourde charge, si le prestataire a pu mesurer l'incompétence de son client, il résistera difficilement à la tentation de gonfler les facturations en exigeant la signature d'avenants au contrat initial pour toute prestation supposée à tort ou à raison ne pas avoir été prévue par le cahier des charges.

Un responsable d'une société de services m'a déclaré franchement que les services publics jouaient fréquemment ce rôle du client incompétent, et que la facturation de services aux administrations n'avait rien à voir avec les tarifs pratiqués pour les entreprises privées. La surfacturation systématique aux dépens des administrations incapables de s'en apercevoir est considérée comme une juste compensation pour des charges sociales exorbitantes et un droit du travail trop rigide.

Si le travail au forfait comporte des inconvénients, le travail en régie n'en est pas exempt. On peut même dire qu'il est pratiquement incontrôlable si le client ne dispose pas de sa propre équipe et d'un chef de projet capable de suivre le travail au jour le jour. Il est tellement confortable d'avoir une réserve permanente de personnels en régie, un peu sous-occupés mais de ce fait assez disponibles en cas d'urgence, et d'emploi plus malléable que les personnels internes ! Cette pratique est à la limite de l'emploi intérimaire, et les clients doivent veiller soigneusement à éviter que les personnels du prestataire ne se retrouvent dans la situation de demander la requalification de leur contrat de travail à leurs dépens.

Pour conclure sur ces aspects de la sous-traitance, il semble raisonnable de dire qu'une sous-traitance modérée peut être un excellent moyen de renforcer le potentiel de l'entreprise, à condition que cette externalisation soit équilibrée par le maintien d'une véritable équipe du client, capable de conserver la compétence interne et de contrôler les prestataires, et pas juste de rédiger des cahiers des charges. L'externalisation à tout crin assortie de la liquidation des compétences internes est une erreur, et celui qui la commet ne tarde pas à être puni par la déliquescence de son système d'information, l'explosion de ses coûts, l'incohérence de ses procédures de gestion, bref par la perte du contrôle d'éléments fondamentaux du fonctionnement de l'entreprise.

L'externalisation peut être un succès si elle est menée par une entreprise compétente qui sait ce qu'elle fait ; elle saura alors choisir de bons prestataires et obtenir d'eux un travail de qualité : à bon chat, bon rat.

L'informatique est pilotée par l'offre

L'informatique est pilotée par l'offre : cette assertion peut sembler immorale, scandaleuse, mais elle est vraie.

Dans le monde idéal dont pourraient rêver les utilisateurs, c'est la demande qui devrait piloter les projets : les maîtres d'ouvrage recueilleraient auprès des stratèges et des responsables fonctionnels l'expression des besoins, qu'ils mettraient sous la forme de cahiers des charges, au vu desquels architectes et maîtres d'œuvre élaboreraient des solutions adaptées. Bref, c'est un refrain que nous avons déjà souvent évoqué et que nous entendrons encore.

Eh bien cela ne se passe pas ainsi. Les stratèges et les responsables fonctionnels n'ont cure d'exprimer leurs besoins sous une forme suffisamment élaborée et raisonnable pour être utilisable, et les cahiers de charges qui résultent de l'écoute de leurs frustrations, si ce n'est de leurs caprices, sont lourds de catastrophes à venir.

L'informaticien face au donneur d'ordres doit adopter l'attitude du médecin face à son patient : les propos de celui-ci sont des symptômes, mais ce n'est pas le malade qui établit le diagnostic ni qui rédige l'ordonnance. Formuler un besoin, c'est dans notre cas rédiger un cahier des charges, dont nous avons dit qu'il serait éminemment modifiable et évolutif. Faire ce travail de rédaction, c'est un métier, le métier d'informaticien, qui rédige en pleine conscience des possibilités techniques, même s'il s'interdit d'effectuer des choix techniques prématurés ou arbitraires.

En fait, c'est l'offre de solutions disponibles qui détermine le caractère raisonnable des besoins, simplement parce que l'évolution technique dans le domaine informatique est très rapide, alors que les structures et l'organisation des entreprises évoluent plus lentement. Il n'est donc pas aberrant de saisir l'occasion de techniques qui arrivent à maturité pour les appliquer à des problèmes que l'on aura mis en évidence justement pour cette occasion.

Cela semble mettre les choses sens dessus-dessous, mais l'observation de cas réels confirme que ce type d'opportunisme donne de meilleurs résultats que le volontarisme qui ferait fi des contraintes techniques. Quant à confier tous les pouvoirs à l'utilisateur final, c'est le moyen le plus sûr d'aller à l'échec en passant par l'explosion budgétaire.

Après l'échec…

Lorsque la taylorisation et l'externalisation aveugles ont échoué il faut chercher autre chose. Les entreprises les mieux dirigées savent que la solution réside généralement dans un savant équilibre entre le maintien d'une compétence et d'un potentiel de développement internes complétés par une sous-traitance introduite en doses bien calculées et aux bons endroits. Mais un tel équilibre ne peut être que provisoire : comme nous l'avons exposé dans l'exemple de la section 2.4.4Comment détruire votre informatiquesection*.17, il viendra forcément un jour où les départements administratifs de l'entreprise auront la peau de l'informatique interne, si elle n'est pas liquidée à l'occasion d'une restructuration ou sur la foi des promesses d'une société de services désireuse de s'implanter. Et les sociétés de service ne sont-elles pas incitées par la logique de leur démarche commerciale à toujours proposer le renouvellement de l'informatique de leurs clients, même si l'utilité de ces innovations est douteuse ? Il serait d'ailleurs vain de prétendre que l'informatique interne n'a aucune part de responsabilité dans son destin, et un prochain chapitre nous montrera pourquoi et comment il est effectivement difficile à l'entreprise d'obtenir de ses informaticiens ce qu'elle est en droit d'en attendre. En tout état de cause, selon la presse professionnelle [92], en cette année 2017, l'espérance de vie professionnelle d'un directeur du système d'information dans une entreprise française privée ou même publique est de l'ordre de vingt mois, ce qui donne la certitude qu'aucun projet cohérent de quelque ampleur ne peut être mené à bien.

Chapitre 5 Méthodes de conception, de spécification et de réalisation

Sommaire

Nous l'avons déjà noté, l'histoire des développements informatiques est scandée par la recherche d'une méthode miracle qui permettrait d'encadrer de façon sûre l'activité de programmation, perçue comme trop coûteuse, voire de la remplacer. Jusque dans les années 1970 on parlait de méthodes d'analyse, en opposant l'analyse à la programmation, puis la terminologie a changé, et l'on parle de méthodes de conception, de modélisation ou de spécification. En gros toutes ces méthodes ont échoué faute d'avoir vraiment compris la nature réelle de cette activité : la programmation n'est pas une fabrication, c'est un acte de conception, et des plus complexes. Cet échec n'est pas tout à fait général, nous citerons dans ce chapitre des méthodes qui ont réussi au moins dans certains cas de figure, mais nous verrons que c'est justement parce qu'elles respectent la nature de la programmation, voire qu'elles sont des langages de programmation déguisés. Parmi les méthodes qui peuvent être créditées d'une aptitude réelle à améliorer la production de logiciels, la méthode B comporte en fait un langage de programmation incorporé, on peut dire qu'elle ne fait que déplacer le niveau d'abstraction de l'activité de programmation sans la remplacer dans son rôle de conception ; *eXtreme Programming* est plutôt un recueil de bonnes pratiques associées à un modèle d'organisation du travail.

Dans le champ de la conception informatique, les métaphores inspirées de l'industrie mécanique ou du btiment sont séduisantes mais fausses, elles ont beaucoup contribué à induire en erreur des managers qui en fait ne demandaient pas mieux, aveuglés par l'agacement considérable suscité chez eux par cette activité de programmation qu'ils ne comprenaient pas, et plus encore par ceux qui la pratiquaient.

Ce chapitre présentera à titre d'exemples un certain nombre de méthodes de conception, sans prétention à l'exhaustivité ni à un classement ; le chapitre suivant exposera quelques problèmes soulevés par le travail avec les informaticiens, population aux caractéristiques un peu particulières.

5.1 Méthodes ancestrales

Les plus anciennes méthodes de conception peuvent être illustrées par SADT dans le monde anglo-saxon ou Merise en France – SADT succédait à SA, et Merise à Warnier. Elles s'inscrivent dans le cadre du cycle de développement en V que nous avons décrit au chapitre 2 et dont nous reproduisons ici le schéma de principe.

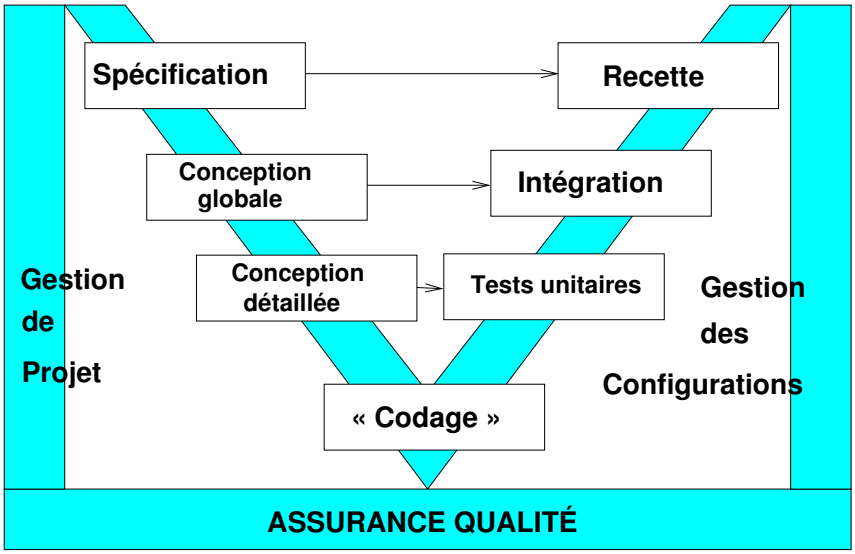

Figure 5.1 : Cycle de développement « en V » du logiciel, inspiré des industries de production matérielle.

Ces deux méthodes reposent sur l'erreur fondamentale selon laquelle l'activité de conception s'arrêterait à la rédaction du cahier des charges et des spécifications détaillées, d'où le texte du programme se déduirait pour ainsi dire mécaniquement, ou en tout cas sans ambiguïté ni complexité majeure. Elles font donc porter l'essentiel de l'effort de leurs adeptes sur la précision et la clarté des documents de définition et de spécification du projet, avec le risque qui en découle (et souvent réalisé) d'un projet sur-spécifié.

Il y a dans ces méthodes, surtout dans Merise, un effort touchant pour éduquer le disciple. Il est vrai que les chefs de projet informatiques n'ont pas forcément été les meilleurs de la classe en philosophie. Essayer de leur enseigner les rudiments d'une pensée systématique n'est donc sans doute pas tout à fait inutile et, s'ils en retiennent quelques leçons tirées du *Discours de la Méthode* de Descartes, l'effort n'aura pas été vain. On constatera d'ailleurs avec étonnement que dans cet exercice les auteurs américains se révèlent souvent de meilleurs disciples de Descartes que ses compatriotes.

Ces méthodes sont assez ennuyeuses, et ont en outre fait la preuve d'une aptitude modérée à atteindre l'objectif qu'elles se proposaient. Nous dirons néanmoins quelques mots de deux d'entre elles, Merise et SADT, parce que, si l'on renonce à construire un modèle complet de l'entreprise, et que l'on se contente d'un usage raisonnable, sans visée totalisante, elles peuvent donner des résultats utiles en facilitant la communication entre acteurs de cultures différentes. Nous pourrions d'ailleurs en dire autant d'à peu près toutes les méthodes disponibles.

5.1.1 Merise

La méthode Merise a été inventée en 1978 par une équipe du CTI (Centre Technique d'Informatique) et du CETE (Centre d'Études Techniques de l'Équipement) d'Aix-en-

Provence animée par Hubert Tardieu, Arnold Rochfeld et René Colletti à la suite d'un appel d'offres du ministère français de l'industrie. Elle se propose pour les activités de conception, de développement et de réalisation de projets informatiques, avec comme objectif la constitution d'un *système d'information*. Elle repose sur la séparation des données et des traitements, et sur la construction de modèles conceptuels, logiques et physiques de données, ordonnés au moyen du *modèle entité-relation*.

La concepteur Merise va entreprendre de construire un modèle fin du fonctionnement de l'entreprise, mais auparavant il lui faut en tracer les grandes lignes. Pour ce faire il va en identifier les *acteurs internes*, ou domaines, puis les flux d'information entre ces domaines. La description de cette charpente de l'entreprise va permettre de travailler domaine par domaine pour construire la représentation détaillée du système tel qu'il existe au départ de la démarche. Armé du modèle entité-relation, le concepteur Merise va décrire le fonctionnement de l'entreprise et de son système d'information.

Le modèle entité-relation

Le formalisme entité-relation est une norme ISO destinée à la description des aspects conceptuels statiques des données pour les systèmes d'information. Il n'est pas propre à Merise et il est utilisé avec d'autres méthodes telles qu'AXIAL ou SADT.

Les *entités* sont des regroupements d'informations, qui pourront par exemple être représentés par une classe du modèle objet et comporter des attributs : ainsi l'entité `être-humain` sera dotée des attributs `nom`, `prénom`, `sexe`, `ge`, etc.

Les *relations* établissent des liaisons logiques entre les entités : ainsi une entité de la classe `être-humain` peut avoir une relation de type `être-propriétaire-de` avec une entité de la classe `automobile`. Une relation peut être factuelle comme `être-propriétaire-de`, ou dynamique comme `acheter`. Une propriété importante des relations est leur *cardinalité* : la relation `être-mère-de` est de cardinalité 1:n (une femme peut être mère de 0, 1 ou plusieurs enfants), cependant que la relation `avoir-pour-mère` est de type 1:1, chaque être humain a une mère et une seule.

Il est bon qu'une entité possède un *identifiant*, c'est-à-dire un attribut ou un groupe d'attributs qui permettent de l'identifier de manière unique. Les lecteurs familiers des bases de données voient que Merise utilise des notions assez voisines de l'algèbre relationnelle, et peut se prêter à la description d'une base de données fondée sur ces principes.

Le conseil souvent donné, et judicieux, pour le choix d'un identifiant, est d'éviter de prendre des attributs significatifs, et au contraire d'attribuer à l'entité un identifiant numérique dépourvue de toute signification. On évitera ainsi le risque de modification d'un attribut servant d'identifiant parce que la réalité sous-jacente aura changé. Ainsi le numéro INSEE, dit « de sécurité sociale », est un mauvais identifiant : un étranger qui arrive sur le sol français est doté d'une immatriculation provisoire et ne connaît son numéro définitif que plusieurs mois plus tard ; et aujourd'hui il y a des personnes qui changent de sexe, phénomène que les statisticiens des années 1940 n'avaient pas envisagé.

Modélisation du Système d'Information

Après avoir recensé toutes les entités de l'entreprise, le graphique obtenu en traçant leurs relations constitue le *Modèle Conceptuel de Données* (MCD), duquel le concepteur Merise va déduire le *Modèle Physique de Données* (MPD) au moyen d'un certain nombre de règles logiques :

Règle n° 1 : toute entité est représentée par une table ;

Règle n° 2 : dans le cas d'entités reliées par des associations de type 1:1, les tables auront le même identifiant ;

Règle n° 3 : dans le cas d'entités reliées par des associations de type 1:n, chaque table possède son propre identifiant, mais l'identifiant de l'entité côté 1,n est recopiée vers la table côté 1,1 et devient un identifiant étranger (index secondaire : chaque individu a comme attribut l'identifiant de sa mère, ce qui permet de retrouver rapidement tous les enfants d'une même femme);

Règle n° 4 : dans le cas d'entités reliées par des associations de type n:m, une table intermédiaire dite table de jointure, est créée, et possède comme identifiant primaire une conjonction des identifiants primaires des deux tables pour lesquelles elle sert de jointure;

Règle n° 5 : etc.

Tout cela semble très satisfaisant pour l'esprit. Il y a juste une objection à formuler : cette activité de modélisation est entièrement informée par l'analyse de l'existant, et pas du tout par l'énonciation des objectifs que le système informatique projeté est destiné à atteindre. La méthode Merise postule que cet objectif se déduit implicitement de la modélisation de la totalité de l'entreprise, ou du domaine concerné de l'entreprise. Nous verrons au chapitre 7 que cette façon d'élaborer un système d'information est utopique, elle suppose que la nature et le sens de l'information stockée par le SI sont des éléments objectifs de la réalité, indépendants du but poursuivi. En fait, comme nous l'apprend la *Field theory of information* dont nous avons parlé à la page 56, une donnée ne devient une information que par l'interprétation qui en est faite, et celle-ci dépendra profondément de celui qui la fait, de ses objectifs et aussi de la date à laquelle il la fait – pour approfondir ces questions on se reportera avec profit aux travaux d'Isabelle Boydens [21] et de Michel Volle [127]. Forts de ces enseignements, nous pensons que la construction d'un système d'information aura quelque chance de succès si sa conception est fonction d'un objectif précis, ce qui veut dire qu'au lieu de recenser toutes les entités que l'on trouve dans l'environnement, il faut au contraire choisir de la façon la plus restrictive possible celles qui vont être utiles. Les adeptes de Merise pourront objecter que rien dans leur méthode ne s'oppose formellement à une telle démarche : sans doute, mais tout dans l'exposé de la méthode et dans la tradition de son usage pèse sur le plateau du « Système d'Information global de l'entreprise », et d'ailleurs le bilan des expériences réelles de l'application de Merise confirme cette tendance.

On aura aussi remarqué que la méthode Merise est fort diserte sur l'architecture des données, mais très discrète sur les traitements. De même, les moyens proposés semblent assez puissants pour modéliser les données, mais beaucoup moins pour ce qui a trait à leurs évolutions dans le temps.

Pour conclure au sujet de Merise, on peut dire que modéliser le fonctionnement de son entreprise est un exercice profitable, à condition de ne pas y passer trop de temps et de ne pas descendre trop dans les détails; dans cette optique une application modérée de Merise peut être utile pour clarifier des processus délicats dont l'importance justifie une telle analyse. Mais soumettre l'ensemble des processus à une telle revue de détail est d'autant plus inopportun que les sommes considérables qui seront englouties dans cette opération n'ont aucune chance de déboucher sur un système d'information global utile, et ce pour les raisons déjà exposées : l'information n'a pas d'existence objective, en soi, elle doit être construite au moment où on en a besoin, et en fonction de ce besoin précis, qui commandera sa nature. Le chapitre 7 expliquera plus en détail pourquoi cette idée de système d'information global est vouée à l'échec. Cette promesse d'échec concerne encore plus les SI « décisionnels » constitués par extraction, à partir des bases de données destinées à la gestion quotidienne, d'informations « stratégiques » à l'usage des dirigeants : non seulement on interprète dans un certains contexte des données élaborées dans un tout autre contexte et pour d'autres motifs, mais en outre on réalise un vaste cocktail de données totalement hétéroclites (voir à ce propos la page 55). Bien sûr aucun véritable dirigeant n'utilise ce genre de chose, mais dans les grands groupes bureaucratisés cela se pratique. On trouvera sur le site de Michel

Volle un exemple [123] qui contredit en partie notre propos : il s'agit de la création, dans une grande entreprise non totalement exempte de bureaucratie, d'un tableau de bord à l'usage du comité de direction ; l'expérience a réussi, mais non sans en passer par un travail très important de ré-élaboration des données, sur lequel nous aurons à revenir.

Alors que des méthodes telles que B ou *eXtreme Programming* sont explicitement dévolues au développement de logiciel, une méthode comme Merise prétend viser un objectif qui serait plus élevé et plus vaste, la conception d'un système d'information. Une telle prétention est largement fallacieuse : concevoir le système d'information n'est d'aucune utilité si l'on échoue à concevoir les logiciels sans lesquels il n'existera pas, or là Merise se révélera dans bien des cas un obstacle plutôt qu'une aide. On pourrait dire de Merise ce que nous avons écrit plus haut d'ISO 9000 : un usage modéré pour des objectifs limités à une vision d'ensemble de l'organisation peut se révéler utile, mais la généralisation systématique pour des usages opérationnels risque d'être catastrophique.

Aujourd'hui René Colletti a modifié son point de vue [85] : après avoir préconisé la modélisation à tout crin du SI, il conseille désormais de le détruire au bulldozer ! Un tel discours est évidemment séduisant pour les directeurs généraux, parmi lesquels la pulsion à la mode est de mettre leur DSI à la porte, si possible accompagné d'une abondante charrette de ses collaborateurs. Colletti se place sur ce marché : selon lui, il faudrait maintenant faire appel à « des techniciens d'audit, qui expertisent les systèmes d'information ; des consultants, qui déterminent ce qui est bon et ce qui ne l'est plus ; des experts des métiers, qui en fixent les enjeux ; des "traducteurs", qui traduiront ces enjeux en termes informatiques, etc. ». Pour ma part je ne trouve aucune des deux positions très convaincante.

5.1.2 *Structured Analysis and Design Technique* (SADT)

SADT[1] est une méthode de modélisation, de conception et de spécification mise au point en 1976 sous la direction de D.T. Ross, de la société américaine SOFTECH, à partir d'une collaboration avec ITT. C'était la belle époque de la programmation structurée, et SADT en est imprégnée.

La méthode ne manque pas de points communs avec Merise, dont elle est à peu près contemporaine, mais elle est dépourvue de son ambition globale et paraît de ce fait plus pragmatique : il s'agit d'aider les concepteurs à modéliser une activité et les entités qu'elle concerne afin de concevoir et de spécifier une application informatique, et il n'y est pas question de système d'information.

SADT pose en principe le cycle de développement en V, que nous avons critiqué à la page 38 : on établit des spécifications détaillées immuables avant d'attaquer la programmation. La méthode partage ici les erreurs issues du parallèle entre l'informatique et les industries de production mécanique ou de construction, et concerne uniquement les étapes qui précèdent le développement.

L'idée est de faciliter la modélisation et la conception par une standardisation d'un certain nombre de documents : tableaux, diagrammes et graphiques, dont la présentation uniforme et systématique codifiée par une véritable *syntaxe graphique* aide les membres de l'équipe de projet à s'y retrouver. De ce point de vue, l'adoption de telles conventions est peut-être judicieuse – mieux vaut une méthode que l'anarchie et le désordre – mais l'expérience tend à prouver que la confection de tels documents est si ennuyeuse que les développeurs ne les mettent pas à jour lorsqu'ils modifient les programmes, qu'ainsi ils deviennent faux, et donc nuisibles.

Il est possible d'utiliser le formalisme entité-relation avec SADT.

La méthode comporte aussi des recommandations en termes de composition et d'organisation de l'équipe de projet, avec notamment une procédure dite *cycle auteur/lecteur*

1 SADT est une marque déposée SOFTECH, et IGL Technology en France.

qui consiste en une transposition aux étapes de spécification de la pratique des revues de code qui existaient déjà (mais sans être assez pratiquées !) pour les étapes de développement. Comme le nom le suggère, il s'agit de soumettre un *kit*, c'est-à-dire un paquet de documents SADT censés représenter la spécification d'une partie de l'application, réalisé par un membre de l'équipe, à l'analyse critique d'autres membres de l'équipe.

Pour conclure au sujet de SADT, si l'on met entre parenthèses le cycle de développement en V, franchement rédhibitoire, il y a sans doute quelques bonnes idées à en retirer, dans la mesure où l'adoption de procédures méthodiques, quelles qu'elles soient d'ailleurs ou presque, introduit un progrès par rapport au désordre spontané, mais rien de décisif pour franchir la barrière du développement logiciel. Bref, ce n'est pas nuisible, mais pas non plus franchement utile.

5.2 Tribut à Frederick P. Brooks Jr.

Frederick P. Brooks Jr fut au début des années 1960 un des principaux concepteurs de l'OS 360, le système d'exploitation des *mainframes* IBM, et le chef de projet de sa réalisation. Il convient de signaler qu'en 2017 le descendant de ce système est toujours utilisé à grande échelle et repose toujours sur la même conception. Il s'agit donc de l'un des plus gros et des plus longs projets de l'histoire de l'informatique, ce qui n'empêche pas son auteur de jeter un regard (auto-)critique sur la plus ambitieuse entreprise d'ingénierie des cinquante dernières années : l'écriture des systèmes d'exploitation.

En 1975 Brooks a écrit un livre [24] intitulé « Le mythe de l'homme-mois », où il tire les leçons d'une vaste expérience qui, de simple programmeur, l'avait conduit à la tête d'un gigantesque projet. En 1995 il en a publié une « édition du vingtième anniversaire », considérablement augmentée de réponses aux objections que lui avait values l'édition initiale, et de nouveaux développements inspirés par l'évolution technique, sociale et intellectuelle de l'informatique pendant cette période.

Quiconque possède la moindre expérience du développement informatique ne peut qu'être frappé par la profondeur et la finesse des observations et des analyses de Brooks, dont la plupart n'ont pas pris une ride malgré les bouleversements du monde informatique. Ce praticien n'a jamais l'œil aussi sûr que pour analyser ses propres erreurs, et l'on constate en le lisant que l'on peut à la fois être un homme de terrain et posséder une vaste culture. Au-delà de quelques phrases lapidaires qui ont frappé ses contemporains (« si une femme peut faire un enfant en neuf mois, neuf femmes peuvent-elles le faire en un mois ? », « lorsqu'un projet a pris du retard, ajouter du personnel à l'équipe ne fait qu'augmenter le retard »), il se livre à une analyse détaillée de l'activité de programmation, de sa nature intellectuelle et de la manière de mener un projet, analyse nourrie de multiples exemples concrets. Il est le précurseur des méthodes modernes que nous évoquerons ci-dessous, et il n'a jamais cru à des inepties comme le cycle de développement en V. En ces temps où les informaticiens réels sortent du tunnel de vingt ans de gestion de projet et de dévaluation systématique de leur activité par des managers dont la morgue n'égale que l'incompétence, il est salubre de relire ce livre pour oser penser que ces méthodes qui leur ont été imposées étaient non seulement désagréables mais ineptes, ce qu'a démontré à l'envi une avalanche de catastrophes dont seul le secret a pu dissimuler l'ampleur. Nous ne saurions bien sûr nous dissimuler que les informaticiens ne sont pas totalement innocents de leurs malheurs, et qu'aux temps de leur splendeur, qui dans certaines entreprises durent encore, ils ont abusé sans vergogne de leur pouvoir, créant par là les conditions d'un retour du bton, selon un expression imagée.

Un point intéressant à noter, c'est que le projet qui constitue la principale expérience de Brooks, l'OS 360, fut aussi un grand chantier d'application des méthodes industrielles à la réalisation de logiciel. Que l'un des principaux responsables du projet en vienne à écrire un livre, à la dimension autocritique explicite et assumée, qui analyse de façon détaillée les

raisons pour lesquelles ces méthodes ne conviennent pas, me semble une contribution de poids à la critique de la taylorisation informatique.

J'ignore s'il a été le premier, mais en tout cas Brooks fut un des premiers à proclamer clairement qu'il est illusoire de prétendre établir au début d'un projet un cahier des charges et des spécifications immuable dont la maîtrise d'œuvre devra assurer docilement la réalisation fidèle. Il explique que le plus difficile dans une telle entreprise consiste à définir le but à atteindre, et que l'on ne peut y arriver que par itérations successives : réalisation d'un prototype sommaire, que l'on montre au donneur d'ordres, qui fait part de ses critiques et de ses suggestions à partir desquelles sera réalisé un second prototype plus élaboré, et ainsi de suite, pendant tout le cycle de vie du système, en fait.

En praticien consommé et en observateur assidu, Brooks a noté que dès lors que l'écriture d'un programme n'est pas effectuée par une personne unique qui en a eu l'idée, mais qu'un groupe de demandeurs s'adresse à un groupe de réalisateurs, les trois quarts du temps cumulé qu'ils vont consacrer ensemble à définir et à construire le programme seront consacrés à des échanges d'information entre eux. C'est notamment pour cela qu'ajouter du personnel à l'équipe d'un projet en retard ne fait qu'accroître le retard : il faut former et informer les nouveaux arrivants, et cela prend du temps qui s'ajoute à celui que les développeurs consacraient déjà à la communication au sein de l'équipe, beaucoup plus de temps que pour l'équipe de balayeurs de couloir auxquels certains managers aiment bien comparer les programmeurs. « Les hommes et les mois ne sont interchangeables que lorsqu'une tche peut être divisée entre plusieurs travailleurs *sans réclamer de communication entre eux*. »

En s'inspirant d'une idée de Harlan Mills, Brooks préconise une organisation de l'équipe de développement selon le modèle d'une équipe chirurgicale dans un bloc opératoire : de même que seul le chirurgien en titre manie effectivement le bistouri, dans une telle équipe seul le chef-programmeur écrit effectivement des lignes de code, ou si d'autres membres de l'équipe en écrivent, c'est sous son étroite responsabilité. Il est assisté d'un adjoint (le « copilote »), pourvu des mêmes compétences que lui, en général plus jeune, avec qui il s'est concerté pendant les phases préparatoires, qui assiste à toutes les phases de son travail, notamment dans les phases de conception, mais s'il écrit du code c'est sous la responsabilité du chef-programmeur. Typiquement, l'adjoint du chef-programmeur le représente dans certaines réunions de projet et peut le remplacer en tant que de besoin, c'est son *alter ego*. L'équipe du chef-programmeur est encore complétée par d'autres fonctions assurées par des spécialistes dont certains ne seront éventuellement pas employés à plein temps, qui pourront contribuer à plusieurs projets, et dont voici l'effectif complet type :

— le chef-programmeur ;
— l'adjoint du chef-programmeur ;
— un administrateur qui s'occupe des tches de gestion (éventuellement à temps partiel pour le projet) ;
— un rédacteur-documentaliste chargé de produire, maintenir à jour et publier les documents relatifs au projet ;
— un responsable des essais, qui prépare les jeux de tests, les exécute et assure les fonctions de recette ;
— un expert du langage de programmation dont le rôle est d'aider l'équipe du projet à résoudre les problèmes techniques de développement (éventuellement à temps partiel pour le projet) ;
— un responsable de l'outillage, c'est-à-dire de l'installation et et de la configuration des logiciels utilitaires nécessaires au projet, tels que système de gestion de configuration ou système de gestion de versions ;
— un archiviste responsable de l'intégration qui assure l'assemblage des différents programmes réalisés par l'équipe ;
— deux secrétaires.

Même si cette organisation paraît fort raisonnable, je doute qu'elle ait été beaucoup utilisée pour des projets informatiques, ce à quoi une collègue me rétorquait un jour qu'il n'y avait que pour l'informatique que cette méthode n'était pas employée, alors qu'il s'agissait du modèle standard d'organisation des équipes de conception et de développement dans tous les autres domaines d'ingénierie. Mais ne s'agit-il pas d'un trait archaïque des industries de fabrication ? Depuis la première édition du livre de Brooks sont apparus de multiples outils informatiques d'aide au développement qui remplacent au moins partiellement des assistants humains : gestionnaires de version, gestionnaires de configuration, outils de documentation et de publication, débogueurs graphiques, etc. Le WWW et les forums en ligne se révèlent de formidables moyens d'assistance technique : je sélectionne avec ma souris le message d'erreur que vient de m'envoyer mon logiciel (compilateur ou autre), je le colle dans la fenêtre de *Google*, et bien souvent j'ai la réponse, voire des dizaines de réponses, avec en prime l'adresse électronique d'experts dont j'ignorais jusqu'à l'existence. Ce n'est pas une démarche intellectuelle très académique, mais elle a le mérite de l'efficacité, avec cette restriction qu'elle fonctionne surtout pour les logiciels libres. C'est ainsi que les informaticiens améliorent leur productivité, et non pas avec des méthodes tayloriennes.

Fort de ses années de pratique, Brooks nous livre une remarque qui pourrait sembler anecdotique, mais qui en réalité touche au cœur de la question : rédiger des cahiers des charges et des documents de spécification et les présenter à des comités de projet au cours d'interminables réunions, c'est extrêmement ennuyeux et démobilisateur. Écrire un programme qui tourne, même s'il ne fait presque rien, est une activité exaltante, non seulement pour le programmeur, mais aussi pour toute l'équipe et pour le maître d'ouvrage, qui se trouvent galvanisés de voir les choses avancer concrètement. Il faut donc réduire au minimum les activités démobilisatrices dans lesquelles le projet va s'enliser, et multiplier les occasions de s'enthousiasmer pour une activité créatrice, dans les limites des objectifs poursuivis bien entendu.

En effet, et c'est en réalité assez mystérieux, mais quiconque a programmé, ne serait-ce qu'un peu, le sait, réussir à faire marcher un programme procure une sensation intense d'allégresse, totalement disproportionnée à l'enjeu réel du programme, qui peut être un exercice minuscule, et en revanche aller se coucher tard dans la nuit sans avoir réussi à résoudre un problème de programmation laisse un arrière-goût de découragement tout aussi excessif. Trouver la solution d'un problème de mathématiques ne procure pas d'émotions aussi violentes. Même des psychologues de la programmation comme Gerald Weinberg [130] n'ont pas vraiment élucidé ce phénomène.

Brooks et Weinberg sont d'accord pour préconiser une « programmation sans ego » : tout acte de programmation est en fait un acte social, puisque tout programme a vocation à être généralisé et partagé. L'abnégation de l'auteur contribuera à rendre ce partage plus facile et plus large. Cette vision évoque une morale un peu naïve, mais on aurait tort de la négliger, elle est effectivement au cœur de l'acte du programmeur, et c'est elle qui explique en grande partie le développement considérable du mouvement du logiciel libre. Et c'est faute de l'avoir comprise que nombre de managers extérieurs au monde de la programmation ont fait échouer des projets informatiques pour lesquels ils croyaient avoir choisi de bonnes méthodes qui n'étaient en fait que des mécanismes autoritaires déguisés mais inefficaces et promptement déjoués par les programmeurs indignés. En termes plus directs, si l'on persécute les programmeurs avec Merise et ISO 9000, on s'expose à un sabotage bien mérité.

5.3 Méthode B

Jusqu'ici notre propos a été plutôt sceptique à l'égard des méthodes systématiques de conception et de spécification, et nous avons eu l'occasion de signaler (page 60) la séparation

presque totale entre le monde des développeurs de logiciel et celui des théoriciens de la preuve de programmes.

Il est des circonstances où le scepticisme doit battre en retraite et où la spécification formelle et vérifiable du logiciel est un impératif indubitable. Nous n'admettrions pas que la validité du logiciel qui pilote les rames de la ligne 14 du métro parisien, dépourvues de conducteur, soit abandonnée au hasard d'une découverte empirique des erreurs. Avoir un accident d'auto parce que le logiciel qui contrôle le dispositif de freinage s'est trouvé dans un état non vérifié nous semblerait inacceptable, et il en va de même pour les avions, les centrales nucléaires et toutes sortes d'autres matériels, dont les systèmes informatiques de pilotage et de contrôle sont en général désignés par la locution *informatique technique*, et que nous avons rangés dans la classe 2 définie par notre introduction page 6. Le présent ouvrage n'est pas vraiment consacré à ce type de systèmes, mais nous décrirons brièvement ici une méthode de construction de programmes qui, sans leur être exclusivement dédiée, est surtout employée à leur intention.

Les systèmes informatiques industriels sont développés dans un contexte très différent de celui qui prévaut pour l'informatique de gestion. Pour les systèmes de gestion (la classe 1 de la typologie des systèmes évoquée dans notre avant-propos page 6) les demandeurs sont des gestionnaires, population généralement peu motivée par la compréhension des problèmes techniques. Il est en outre frappant de constater, dans le domaine de la gestion, que, si les équipes du maître d'ouvrage sont souvent insuffisamment compétentes – ce qui n'est guère surprenant[2] – les équipes des sociétés de service qui interviennent sont constituées de personnels dont les diplômes en informatique sont pour le moins modestes, voire inexistants. En informatique industrielle demandeurs et réalisateurs sont tous des ingénieurs, ce qui imprime à leurs relations mutuelles un caractère très différent de ce que l'on observe en informatique de gestion. On se rappellera une autre différence, déjà signalée par l'encadré p. 35 : les artefacts techniques pilotés par les systèmes informatiques de classe 2 disposent, pour que leur correction puisse être vérifiée, d'un référentiel solide : les lois de la physique ; les systèmes de classe 1 en sont dépourvus.

Si de surcroît le système informatique industriel à réaliser comporte des propriétés critiques, comme dans les exemples cités ci-dessus, les budgets et les délais vont être d'un autre ordre de grandeur, ce qui étend le champ des possibilités techniques[3].

La méthode B [2, 10] a été inventée par Jean-Raymond Abrial au milieu des années 1980 pour développer des logiciels prouvés. Elle a été utilisée dans le cadre du projet de métro sans conducteur METEOR réalisé par Matra (maintenant *Siemens Transportation System*) pour le compte de la RATP, pour construire de façon sûre les parties du logiciel qui jouent un rôle critique pour la sécurité des passagers. La partie du logiciel du métro METEOR réalisée grâce à l'Atelier B (l'outil informatique sous-jacent à la méthode [30] développé par la société ClearSy) comprend près de 100 000 lignes de code Ada générées automatiquement. Notons qu'avant B Abrial avait créé le langage de spécification Z.

Les premières démarches de preuve de programme tentaient d'appliquer des procédures de preuve à des programmes déjà construits. Il s'est assez vite révélé qu'un programme final était un objet beaucoup trop complexe pour être soumis d'un seul coup à une procédure de preuve, manuelle ou à plus forte raison automatique. L'idée de B est donc d'élaborer

2 Il n'y a qu'en France – et dans quelques pays qui partagent son sous-développement informatique – que l'on considère comme normale la situation d'analphabétisme informatique profond qui affecte les gestionnaires et dirigeants d'entreprise. On pourra consulter un texte déjà cité de Michel Volle[118], qui semble ouvrir la possibilité d'étendre ce diagnostic à d'autres domaines techniques qui concernent l'entreprise.

3 Lors du dîner des *Journées Francophones des Langages Applicatifs* (JFLA, `http://jfla.inria.fr/`) 2010 je me trouvai à la table de l'équipe Inria qui développe le système Coq de preuve de logiciel, ce qui me donna l'occasion de leur demander quel était, à leur avis, le ratio à appliquer au temps de développement d'un logiciel ordinaire pour calculer le temps de développement d'une version *prouvée* du même logiciel. Après une brève concertation leur réponse fut « 100 » (cent). Voilà de quoi dissiper bien des illusions.

la preuve en même temps que le programme. Le langage de développement B permet de spécifier d'une part le programme proprement dit, d'autre part les propriétés que l'on souhaite lui voir respecter. Le manuel de référence de l'atelier B, disponible sur le site, donne l'explication suivante :

« L'écriture d'un programme B consiste à rédiger les spécifications formelles de machines abstraites à l'aide d'un formalisme mathématique de haut niveau. Ainsi, une spécification B comporte des données (qui peuvent être exprimées entre autres par des entiers, des booléens, des ensembles, des relations, des fonctions ou des suites), des propriétés invariantes portant sur ces données (exprimées à l'aide de la logique des prédicats du premier ordre), et enfin des services permettant d'initialiser puis de faire évoluer ces données (les transformations de ces données sont exprimées à l'aide de substitutions). L'activité de preuve d'une spécification B consiste alors à réaliser un certain nombre de démonstrations afin de prouver l'établissement et la conservation des propriétés invariantes en question (par exemple il faut prouver que l'appel d'un service conserve bien les propriétés invariantes). La génération des assertions à démontrer est complètement systématique. Elle s'appuie notamment sur la transformation de prédicats par des substitutions. »

On aura compris que B est un système de développement complet, qui comporte son propre langage de programmation, ce qui confirme en fait l'idée qu'une méthode de spécification est soit un langage de programmation, soit inutile, et que de toute façon la création d'un système informatique demande une vraie compétence en programmation. Il n'y a pas de magie possible. Il semble clair que les exigences de B en termes de délais et de qualification technique du personnel sont relativement élevées par rapport à du développement classique de logiciel non critique. Le tout est de ne pas se tromper dans la détermination de ce qui est critique et de ce qui ne l'est pas.

Le développement avec l'atelier B des 100 000 lignes de code Ada critique pour le logiciel de la ligne 14 du métro parisien (sans pilote à bord) a engendré 30 000 obligations de preuve, dont plus de 90% ont été réalisées automatiquement. Les quelque 2500 preuves qui ont résisté aux procédures automatiques ont nécessité plusieurs mois de travail humain, mais l'industriel a estimé que le bilan était largement positif grce à l'économie engendrée par la suppression des tests de bas niveau. Après quelques années d'exploitation sans incident notable, la conclusion s'impose que la méthode B est efficace et sûre pour les développements critiques.

Pour être complet il faut également signaler les limites de la méthode B :

— les capacités de preuve sur des formules comportant des opérations arithmétiques sont limitées ;
— le développement formel de systèmes contenant des calculs numériques n'est actuellement pas possible avec la méthode B et les outils associés ; de tels calculs restent sous-spécifiés et les preuves de correction ne peuvent être données ;
— absence de vérification de propriétés temporelles due à la logique supportée (par atelier B) ;
— on ne peut pas décrire avec B les phénomènes concurrents, les fenêtres de temps et plus généralement le temps réel (codage des événements par des variables) ; les logiques temporelles sont plus adaptées pour spécifier le comportement dynamique.

5.4 *Unified Modeling Language* (UML)

UML est un langage de modélisation [86] et de conception de systèmes informatiques, plus particulièrement adapté à la programmation par objets (voir section4.2.5La programmation par objetssection*.23). Il est né en 1997 du rapprochement de plusieurs méthodes inventées par de fortes personnalités : Grady Booch [17], qui s'était illustré de façon brillante dans le domaine des développements en Ada avant d'élargir son activité à d'autres langages,

Jim Rumbaugh, auteur principal d'OMT *(Object Modeling Technique)* et Ivar Jacobson, créateur des cas d'utilisation *(use cases)* et d'OOSE *(Object-Oriented Software Engineering method)*.

Ce qui a déclenché l'éclosion entre la fin des années 1980 et le milieu des années 1990 d'une cinquantaine de méthodes de modélisation par objets, dont UML est un aboutissement, c'est la constatation du hiatus entre les avantages techniques de la programmation par objets, qui ne sont plus guère contestés, et les difficultés de sa mise en œuvre. En effet, aussi étrange que cela puisse paraître, les développeurs éprouvent plus de difficultés à analyser un problème informatique en termes d'objets qui interagissent qu'à le décomposer en fonctions et en données élémentaires. Or l'approche objet nécessite une grande rigueur dans la description de l'univers à modéliser et dans l'emploi des concepts, faute de quoi le risque d'échec est grand – cela est vrai de toute approche de la programmation, mais se manifeste d'autant plus avec la démarche par objets que celle-ci se veut plus exigeante intellectuellement. Et c'est pour pallier l'absence de méthodologie standard de modélisation d'une application avec des objets qu'ont été élaborées les méthodes qui ont finalement convergé pour donner UML. En prime, UML offre bien sûr l'espoir sans cesse renaissant de permettre enfin une division du travail réussie entre concepteurs (du modèle) et réalisateurs (du logiciel) : le lecteur parvenu à ce point de notre ouvrage ne sera pas surpris d'apprendre que de ce côté-là, pour les raisons fondamentales déjà évoquées à de nombreuses reprises, la déception risque d'être de mise, mais cela ne retire pas forcément toute utilité à UML.

UML, normalisé par l'OMG *(Object Management Group)*, est à la fois un langage de modélisation, un support de communication et un cadre méthodologique. C'est un langage formel défini par un métamodèle qui décrit de manière précise tous les éléments de modélisation (les concepts véhiculés et manipulés par le langage) et la sémantique de ces éléments (leur définition et le sens de leur utilisation).

UML se présente comme un métalangage par rapport aux langages de programmation, il repose sur une notation graphique indépendante des langages de programmation qui donne à chaque concept objet une représentation particulière, au moyen de neuf types de diagrammes qui sont autant de vues mutuellement cohérentes sur le même modèle :

- les quatre diagrammes structurels, qui donnent de l'application une vue statique :
  - diagramme de classes,
  - diagramme d'objets,
  - diagramme de déploiement,
  - diagramme de composants ;
- les cinq diagrammes comportementaux qui donnent une vue dynamique de l'application :
  - diagramme d'activités,
  - diagramme de séquence,
  - diagramme d'états-transition,
  - diagramme de collaboration,
  - diagramme de cas d'utilisation.

La confection de ces diagrammes[4] au moyen d'un outil UML adéquat tel que *Rational Rose* permet de générer automatiquement le code qui correspond aux diagrammes, au choix

4 Les diagrammes sont devenus treize depuis UML 2.0 paru en 2002, mais nous en resterons à la version 1 ; la version 2.0 est une tentative pour englober le monde du temps réel, mais à mon avis cela ne fait qu'introduire de la confusion dans un système déjà complexe. Le diagramme de collaboration a été rebaptisé « de communication » et les diagrammes supplémentaires sont :
   - diagramme de structure composite ;
   - diagramme de vue d'ensemble des interactions ;
   - diagramme de paquetage ;
   - diagramme d'échéances.

en Java, C++ ou Visual Basic. Nous voici donc ramenés à un langage de programmation graphique.

Un langage de programmation graphique soulève deux types d'objections :

— Comme nous l'a appris Brooks, un logiciel est par nature invisible, et les tentatives pour le dessiner sont toujours incomplètes. Il faudrait pour y arriver plusieurs niveaux de graphiques :
  — le graphique de l'organisation statique du programme en classes et en méthodes, effectivement présent dans UML sous la forme des diagrammes de classes, d'objets et de composants ;
  — le graphique temporel du flot de données, censément présent dans UML, mais sous la forme déjà beaucoup moins convaincante des diagrammes de séquence ;
  — le graphique des références croisées de passage de paramètres entre différentes parties du programme, ou de passage de messages entre objets, bien difficile à représenter de façon complète, et qui laisse de côté la question des valeurs possibles des données ainsi échangées, partiellement représenté en UML par les diagrammes de collaboration ;
  — le graphique des structures hiérarchiques de données ;
  — et d'autres encore, comme en UML les diagrammes d'états-transitions ou d'activités, pour aboutir à la conclusion que la superposition de tous ces diagrammes serait un enchevêtrement illisible sans pour autant nous donner une vision complète du logiciel.
— Les représentations de haut niveau des programmes, qu'il s'agisse de diagrammes ou de texte en pseudo-code inspirés d'un langage de programmation réel, comme on en trouve souvent dans les livres d'algorithmique, masquent les détails de la programmation, tels que les bornes exactes de l'intervalle de variation d'un indice, par exemple, et le traitement à appliquer aux cas limites déclenchés par l'arrivée à une de ces bornes. Or tous les programmeurs le savent, c'est justement là que se commettent les erreurs de programmation. C'est mesquin mais c'est ainsi. Un langage de programmation qui escamote ces cas risque donc d'être assez imprécis, ce qui est assez regrettable. Nous craignons que ce ne soit le cas d'UML.

  Il faut noter qu'UML est flanqué d'un langage de programmation textuel, OCL *(Object Constraint Language)*, qui comporte notamment des opérations sur les collections d'objets qui permettent de lever la restriction énoncée ici, mais au prix de l'abandon des avantages supposés de la programmation graphique : cela nous semble paradoxal ; à partir du moment où on est contraint d'apprendre OCL pour être précis, autant programmer directement en Java.

Les vrais développeurs ont le plus souvent tendance à considérer UML comme une gêne, une entrave et un moyen de produire des documents qui ne seront pas mis à jour, et qui seront donc faux et de ce fait nuisibles.

Reste à résoudre un problème : lorsqu'un demandeur s'adresse à un réalisateur pour lui demander de réaliser un logiciel, il faut bien trouver un moyen de décrire ce logiciel. Il est possible de considérer UML comme une étape de la programmation, antérieure au « codage » proprement dit : il s'agit de bien classer ses idées avant de commencer à coder – ce qui n'exclut pas, bien sûr, les itérations ultérieures face aux difficultés du codage. De ce point de vue, la modélisation est un moment de l'activité de programmation, elle est effectuée par les mêmes personnes que le codage, mais lors de périodes de temps différentes.

Un usage modéré et non dogmatique d'UML pourra peut-être aider au démarrage d'une activité de conception ; la méthode *eXtreme Programming* examinée à la page 89 propose des solutions peut-être complémentaires ou plus prometteuses.

Ne quittons pas UML sans mentionner son dernier défaut (mais non le moins gênant) : c'est, comme nous l'avons dit, une norme publiée par l'OMG, et pour avoir accès au dernier

état de la norme il faut être membre de l'OMG, c'est-à-dire payer une cotisation. Nous aurons l'occasion d'y revenir plus loin dans ce chapitre, faire payer l'accès à une norme est contre-productif, cela ne fait que dissuader les gens de s'y conformer. Rendre un outil de développement payant, et *a fortiori* cher, c'est en détourner l'immense population de développeurs indépendants, de PME et d'étudiants qui sont aujourd'hui la force vive du développement logiciel.

5.5 Méthodes agiles : *eXtreme Programming*

La méthode *eXtreme Programming* [9] (XP en abrégé) se situe dans la lignée des méthodes de développement rapide ou de prototypage rapide tout en s'en distinguant, on peut dire aussi qu'elle est l'héritière des idées de Brooks, que nous avons rappelées plus haut. Elle a été créée à la fin des années 1990 par Kent Beck, Ward Cunningham et Ron Jeffries. Les auteurs du livre cité en référence l'ont introduite en France au début de notre siècle. Cette méthode et d'autres assez semblables sont maintenant appelées « méthodes agiles ».

Un développement conforme à la méthode *eXtreme Programming* (nous dirons « développement XP ») obéit à un cycle en V qui paraît assez semblable à celui que nous avons critiqué plus haut, mais au lieu de partir d'un cahier des charges monumental et immuable et de parcourir une fois le cycle en deux, trois ou quatre ans, on part d'une définition d'objectif très générale et laconique, en fait des *scénarios* établis par le client, et on va parcourir de nombreuses fois le cycle au cours d'itérations dont chacune pourra durer un laps de temps aussi bref que deux ou quatre semaines. Chaque itération amènera une définition plus affinée de l'objectif et un accroissement des capacités du système, étant entendu que dès la fin de la première itération le système sera opérationnel, même s'il ne fait à peu près rien. La durée de l'itération doit être fixée au départ ; une fois qu'elle est fixée il est important de la respecter et de procéder à chaque fin de cycle aux opérations de réception des travaux accomplis pendant cette période. L'intérêt de la méthode, on le comprend, réside dans la définition progressive mais néanmoins rigoureuse de l'objectif, et dans la capacité à s'adapter aux changements de cap en cours de navigation.

eXtreme Programming repose sur un certain nombre de *pratiques XP* canoniques :

— la programmation en binôme : tout le travail de programmation est effectué à deux programmeurs par poste de travail, un qui écrit et l'autre qui regarde ; la composition des binômes varie fréquemment, programmeurs expérimentés et débutants échangent régulièrement leurs places ; ainsi tous les membres de l'équipe connaissent le code de toutes les parties du projet et les débutants profitent de l'expérience des experts ;

— les tests unitaires : toute écriture d'une partie de l'application commence par la conception et l'implémentation des tests qui permettront de la valider ;

— la conception simple : on ne cherchera pas à anticiper les complications futures éventuelles, chaque fragment de code est rédigé de la façon la plus simple possible compte tenu des spécifications du moment ;

— le remaniement : il s'agit de la mise en facteur commun et de la généralisation de parties du code qui correspondent à des motifs *(patterns)* récurrents ;

— la responsabilité collective du code, induite et facilitée par la programmation en binômes de composition variable ; on note ici une divergence avec l'équipe du chef-programmeur de Brooks : les temps ont changé et l'autorité du chef n'est plus acceptable, du moins si elle est explicite ;

— les règles de codage, destinées notamment à assurer l'homogénéité du code, d'autant plus nécessaire que chaque programmeur doit pouvoir intervenir sur chacune de ses parties ;

— l'intégration continue permet la livraison d'un système opérationnel à la fin de chaque itération du cycle de développement, dont on rappelle que la durée est fort brève (deux à six semaines sont des valeurs habituelles) ;
— le rythme durable : les « charrettes » hystériques pour livrer un projet en retard sont prohibées et remplacées par un rythme de travail régulier.

À ces préceptes il convient d'ajouter un style particulier de relation avec le client. Précisons tout de suite cette notion de client : idéalement ce sera le vrai client, celui qui paie, mais comme la méthode XP suppose sa présence assidue au sein de l'équipe de développement et sa participation active tant à la rédaction des scénarios qu'à la réalisation des tests de recette, dans la réalité il ne sera pas toujours possible de satisfaire à cette condition, et le rôle du client pourra par exemple être tenu par ce que l'on appelle en gestion de projet la maîtrise d'ouvrage déléguée[5]. La méthode précise les droits du client :

— disposer d'une vue d'ensemble du projet, d'un calendrier et d'un budget prévisionnels ;
— en avoir pour son argent ;
— juger des progrès sur un système opérationnel, selon les tests reproductibles qu'il aura rédigés (ou fait rédiger) ;
— pouvoir changer d'avis, modifier les spécifications et les priorités sans s'exposer à des augmentations de budget exorbitantes ;
— être informé des dépassements de délais suffisamment tôt pour pouvoir réduire le périmètre fonctionnel afin de retomber sur la date de livraison initialement prévue ;
— interrompre le projet à tout moment, et disposer néanmoins à cet instant d'un système utile et utilisable conforme aux sommes investies jusque-là.

XP suppose un travail d'équipe très fusionnel : tous les membres de l'équipe, y compris le client, doivent être dans le même bureau, où les postes de travail seront disposés de telle sorte que chacun puisse voir l'écran de chacun (les manuels donnent le plan conseillé pour le bureau).

La méthode définit aussi des *rôles XP* :

— programmeur ;
— client ;
— testeur ;
— traqueur (celui qui tient le tableau d'avancement des travaux) ;
— manager ;
— coach.

Nous avons déjà défini le rôle du client ; s'il doit cumuler son rôle avec un autre, ce ne peut être que celui de testeur.

Le manager est en fait généralement extérieur à l'équipe proprement dite. Il est le supérieur hiérarchique des programmeurs et peut leur demander des comptes, notamment sur les calendriers et les budgets. S'il doit ajouter un autre rôle au sien, ce ne peut guère être que celui de traqueur.

Le coach, à l'inverse du manager, appartient pleinement à l'équipe, il en est même la cheville ouvrière, il veille à ce que chacun joue bien le rôle qui lui est dévolu et respecte les pratiques XP, il aide le client à rédiger des scénarios.

La littérature XP ajoute à ces principes un adage moins formel mais important néanmoins : le principe « diviser pour régner » est d'une grande efficacité face à un ennemi, il est contre-productif et totalement hors de propos vis-à-vis de sa propre équipe.

On l'aura compris, XP est moins une méthode de conception au sens habituel du terme qu'une méthode d'organisation du travail de conception. Les principes XP ne peuvent

5 Dans l'appellation « maîtrise d'ouvrage déléguée », c'est le client qui délègue, et pour bien faire il importe que cette MOAD fasse partie de son organisation.

que séduire quiconque a une expérience de la programmation et du développement, parce qu'ils constituent des solutions rationnelles aux problèmes bien connus de cette activité, et parce qu'ils formulent implicitement une critique radicale des méthodes punitives et improductives du passé.

On peut craindre que la mise en pratique des principes XP, qui supposent abnégation (« programmation sans ego ») et rigueur sans faille, ne se heurte à la sociologie réelle des populations concernées, notamment dans un pays latin comme la France où les signes extérieurs du pouvoir et du prestige ont souvent au moins autant d'importance que la satisfaction intériorisée du devoir accompli.

Une objection souvent formulée à l'encontre des principes XP, c'est qu'ils ne seraient adaptés qu'à des projets de petite taille. Mais justement, une des conclusions auxquelles la rédaction de ce livre nous a conduit, c'est que la grande taille de beaucoup de projets n'est pas vraiment justifiée par l'objectif poursuivi, et qu'il faut essayer de réduire la taille des projets dans toute la mesure du possible. C'est très souvent possible.

5.6 Le modèle du logiciel libre

5.6.1 Définition

Après avoir mobilisé les esprits et les langues, le mouvement du logiciel libre est aujourd'hui (en 2017) un phénomène économique notable dont les acteurs sur la scène informatique ne peuvent plus ignorer l'existence. Rappelons-en brièvement les principes.

La locution « logiciel libre » désigne les logiciels produits sous une licence dite *libre* parce qu'elle garantit à tous et à chacun *quatre libertés* caractéristiques :

— liberté d'utilisation ;
— liberté de copie ;
— liberté de modification ;
— liberté de redistribution.

Le dispositif établi par cette règle des quatre libertés est bien plus fort que la simple gratuité de logiciels tels que les *freewares* : il fait du logiciel ainsi déclaré libre un *bien public*.

D'un point de vue économique, le logiciel présente en effet des traits d'un bien public et des traits d'un bien privé :

— le coût de production pratiquement engagé en totalité dès le premier exemplaire, l'usage non destructif (il peut être utilisé par un nombre infini d'utilisateurs), l'usage non exclusif (il est difficile d'empêcher quelqu'un d'autre de l'utiliser) sont des caractéristiques d'un bien public ;
— le recours à la protection du droit d'auteur ou du brevet permet d'annuler les aspects « publics », par exemple en limitant la reproductibilité, et de faire du logiciel un bien privé.

L'alternative se situe entre le logiciel comme bien privé, idée des entreprises commerciales, et le logiciel comme bien public, idée du logiciel libre.

Il découle de ces principes que le statut du code source détermine la nature publique du bien. La publicité du code interdit l'appropriation privée.

Outre le modèle économique exposé ci-dessus, le logiciel libre est aussi un mouvement associatif créé en 1984 par Richard M. Stallman sous le nom de *Free Software Foundation (FSF)*. Stallman, informaticien au MIT et auteur depuis 1975 d'un logiciel extraordinaire, Emacs, a aussi créé en 1983 le projet *GNU (« GNU is Not Unix »)* destiné à créer un système d'exploitation libre de droits et dont le texte source serait librement et irrévocablement disponible à tout un chacun pour l'utiliser ou le modifier.

La principale licence libre (mais ce n'est pas la seule) est la *GNU GPL (GNU General Public License)*, qui impose ses mêmes termes à tout logiciel dérivé d'un logiciel sous licence GPL.

5.6.2 Le logiciel libre n'est pas un paradoxe

Le comportement de ceux qui achètent même cher des choses disponibles gratuitement n'est pas forcément aussi aberrant qu'il semble au premier abord : récupérer un logiciel sur le réseau, l'installer, le mettre en œuvre, suivre la sortie des nouvelles versions, s'assurer de son fonctionnement correct et régulier, fournir de l'assistance technique aux utilisateurs demande un travail non négligeable, et payer des gens pour le faire à votre place est un choix qui peut être raisonnable.

Le développement d'un logiciel consistant demande plusieurs années. S'il faut n mois pour écrire un programme destiné à traiter un problème pour mon usage personnel, l'adapter à une utilisation plus générale par des gens que je ne connais pas nécessite des modifications et l'écriture d'une documentation dont on estime en général l'ordre de grandeur du coût à $10n$ mois. Si maintenant je veux vendre ce logiciel dans les grandes surfaces, ce qui va coûter le plus cher c'est le marketing, la publicité, la constitution du réseau de distribution, l'emballage et la logistique, répondre au téléphone aux clients insatisfaits. Cela coûte (pour commencer) $100n$ mois de travail. Bref le travail de conception initial représente 1% du travail total, le travail technique au sens large 10%. Clairement ce n'est pas là que va porter l'effort principal de l'éditeur commercial.

Incidemment une conséquence de cet état de faits (dont ceux qui n'ont jamais écrit de logiciel ont peu conscience) est que la commercialisation d'un logiciel ne présente d'intérêt que si le marché espéré est considérable soit par le nombre d'unités vendues (cas de *Microsoft Word*) soit par le prix astronomique (cas de certains logiciels de modélisation moléculaire). Sinon il vaut mieux soit le diffuser comme logiciel libre, soit en céder les droits commerciaux à une entreprise dont c'est le métier, ou les deux puisque ce n'est pas incompatible. Ces faits élucident aussi le paradoxe apparent de l'existence simultanée de versions libres et commerciales du même logiciel : l'objet technique n'est pas la partie la plus coûteuse du logiciel vénal.

5.6.3 Le développement du logiciel libre

Le modèle du logiciel libre n'est pas sans influence sur la nature même du logiciel produit. En effet, dans ce contexte, un auteur peut facilement utiliser d'autres logiciels s'ils sont libres, que ce soit pour les utiliser directement tels quels au sein de son propre programme, pour recourir à des bibliothèques de fonctions générales ou pour s'inspirer d'un logiciel aux fonctions analogues mais dans un environnement différent.

Des systèmes de développement coopératif se mettent en place par le réseau, qui seraient impensables pour du logiciel non libre : les programmes sous forme source sont accessibles sur un site public, et chacun peut soumettre sa contribution. L'archétype de ce mode de développement est celui du noyau Linux proprement dit, coordonné par Linus Torvalds personnellement ; ce travail mobilise, pour 3 800 000 lignes de code, plus de 300 développeurs qui se connaissent souvent uniquement par courrier électronique, bien que des réunions plénières soient organisées une fois par an.

Pour qu'un tel procédé donne des résultats utilisables, il faut que le logiciel présente une architecture qui s'y prête, notamment une grande modularité, afin que chaque contributeur puisse travailler relativement indépendamment sur telle ou telle partie.

Finalement, la réutilisation de composants logiciels, dont plusieurs industriels parlent beaucoup depuis des années sans grand résultat, sera sans doute réalisée plutôt par les adeptes de l'Open Source. En effet, l'achat d'un composant logiciel est un investissement problématique, tandis que le récupérer sur le réseau, l'essayer, le jeter s'il ne convient pas, l'adopter s'il semble prometteur, c'est la démarche quotidienne du développeur libre. On pourra lire à ce sujet l'article de Josh Lerner et Jean Tirole, *The Simple Economics of Open Source* [74].

Dans le monde du libre le logiciel commence à prendre forme autour d'un noyau, noyau de code et noyau humain, généralement une seule personne ou un tout petit groupe. L'impulsion initiale procède le plus souvent du désir personnel des auteurs de disposer du logiciel en question, soit qu'il n'existe pas, soit que les logiciels existants ne leur conviennent pas, que ce soit à cause de leur prix ou de leur environnement d'exécution. Puis d'autres contributions viennent s'ajouter, presque par accrétion. Un coordonnateur émerge, souvent l'auteur initial : il assure la cohérence de l'ensemble. Quand des divergences de vues surgissent, il peut y avoir une scission *(forking)*.

Les outils de travail des développeurs du libre sont (outre les outils de développement proprement dit tels qu'éditeurs de texte, compilateurs et débogueurs) le courrier électronique, des serveurs WWW pour abriter les sites d'archivage du logiciel tels que `https://sourceforge.net/` ou `https://github.com/` et des outils de gestion de version tels que CVS *(Concurrent Versions System)* destinés, comme le nom l'indique, à permettre le développement en réseau et à plusieurs d'un logiciel commun.

Le logiciel libre a pris son propre style, qui n'est pas sans points communs avec les principes énoncés ci-dessus de l'*eXtreme Programming* : pas de représentation graphique (cela passe mal par courrier électronique), pas de commentaires, la simplicité de l'implémentation est considérée comme plus importante que celle de l'interface. On consultera avec intérêt à ce propos les *GNU Coding Standard* [104], qui associent d'excellents conseils de programmation à de moins bons et à des injonctions morales et idéologiques parfois surprenantes. La lecture de ce document montre combien le monde du logiciel libre est imprégné des idées et des pratiques de la réutilisation de composants ou de logiciels déjà disponibles. Si on compare ces pratiques à celles du monde non libre, comme par exemple les EJB [108] *(Enterprise JavaBeans)*, destinés eux aussi à la réutilisation, on est frappé du contraste entre l'efficacité rustique des méthodes « libres » et la complexité qui se dégage des 646 pages de spécifications des EJB [36].

Un article [43] de Richard Gabriel résume assez bien la vision générale qui est à la base des méthodes et pratiques de conception des développeurs du logiciel libre par quatre principes énoncés dans leur ordre de préséance :

— Simplicité de conception, pour l'implémentation comme pour l'interface. C'est le plus important ; la simplicité de l'implémentation passe avant celle de l'interface.
— Justesse : la conception ne doit pas comporter d'erreur patente.
— Cohérence : la conception ne doit pas être incohérente, même si une part de cohérence peut être sacrifiée à la simplicité.
— Complétude : la conception doit prévoir toutes les situations importantes, dans les limites du raisonnable.

Ce qui est important dans ces principes, c'est bien sûr la hiérarchie entre eux, et surtout la prééminence de la simplicité à laquelle même la justesse peut être sacrifiée. En d'autres termes, il est prescrit de ne pas doubler la taille et la complexité d'un logiciel pour qu'il comporte la solution parfaite d'un cas de figure à l'occurrence exceptionnelle. Il est clair qu'un système d'exploitation comme VMS (des ordinateurs VAX de Digital Equipment) ou un langage de programmation comme Ada sont devenus obèses pour avoir voulu implanter des solutions parfaites à des problèmes qui n'intéressaient pas grand monde, et ont ainsi perdu des qualités de simplicité et de sobriété qui auraient plu à tout le monde. La lourdeur et la complexité de ces logiciels les ont rendus trop chers, trop encombrants, trop lents, trop difficiles à comprendre. À l'inverse, les versions initiales d'Unix et de C, qui n'étaient pas des logiciels libres mais qui en ont formé la matrice, implantaient des solutions d'une simplicité déconcertante, souvent au prix d'une certaine insuffisance conceptuelle, mais rétrospectivement il est patent que cette approche a donné de meilleurs résultats.

Comme Richard Gabriel le souligne, le langage C a été conçu pour faciliter l'écriture de son compilateur (le programme qui traduit en langage machine les programmes écrits par les utilisateurs), et c'est au programmeur de se plier à une syntaxe franchement atroce pour

écrire du code acceptable par ce compilateur. L'application de la règle de simplicité maximum se fait donc aux dépens de toutes les qualités de clarté, de lisibilité et d'intelligibilité prônées jadis par les adeptes de la programmation structurée, mais en contrepartie le langage C (et le système Unix pour l'écriture duquel il a été inventé) ont gagné une versatilité et une souplesse de mise en œuvre qui leur ont assuré le succès que l'on sait.

À la fin de son article Richard Gabriel énumère comme un repoussoir la recette du « grand système complexe » : pour commencer définir les spécifications de la « chose correcte » ; puis définir son implémentation ; enfin l'implémenter. Comme c'est la « chose correcte », 100% ou presque des caractéristiques requises sont là, mais comme la simplicité d'implémentation n'a jamais été une priorité, c'est long à implémenter. C'est grand et complexe ; pour l'utiliser il faut des plate-formes et des outils gros et chers.

Vous ne pouvez utiliser le « grand système complexe » que si votre employeur est riche : si son auteur a voulu créer un logiciel pour utilisateur final, c'est peut-être un choix commercial judicieux mais, si c'est un outil de développement, l'inconvénient est exponentiel, parce que de nos jours le développeur est souvent quelqu'un qui aime travailler la nuit chez lui, quand ce n'est pas un étudiant, alors si votre système de développement est cher et ne fonctionne que sur des plate-formes chères, il n'y aura pas beaucoup de logiciel développé avec lui. C'est ce qui est arrivé par exemple au langage Ada, malgré ses qualités éminentes : pendant longtemps ce fut un langage de riches, parce que sa conception procédait d'un appel d'offres du DoD, et que les clients étaient les contractants du DoD et du secteur aérospatial, à la réputation méritée de prospérité ; mais de ce fait le monde universitaire était laissé au bord de la route, ce qui se révéla une erreur fatale pour la dissémination du langage. Depuis quelques années un excellent environnement de développement libre est disponible (GNAT), mais pour permettre la popularité du langage c'est peut-être trop tard.

Toujours dans le domaine des outils de développement, on peut observer un contre-exemple au cas Ada : c'est le succès du système *Eclipse*, logiciel libre créé par IBM pour le développement de programmes Java. Cette publication sous licence libre a conféré à *Eclipse* un succès rapide, a tel point qu'il a assez vite supplanté les systèmes commerciaux. Une autre qualité d'*Eclipse* réside en ceci qu'il est facile de lui ajouter des extensions ou des adaptations par de petits programmes externes *(plugins)* : ont ainsi vu le jour des adaptations à d'autres langages, ainsi qu'à des méthodes particulières, comme *eXtreme Programming*. Cet exemple vient à l'appui de l'idée que les outils de développement ou d'infrastructure ont tout intérêt à être libres, même si leur créateur semble au départ perdre des bénéfices, parce que ces outils permettront la création ultérieure de plus de logiciels dans des conditions économiques plus avantageuses. On conçoit que Microsoft hésite à renoncer aux 70% de marge d'exploitation de Windows, mais peut-être les rattraperait-il facilement avec une version d'Office pour Linux...

Le logiciel libre n'est peut-être pas la panacée de tous les problèmes informatiques, mais il a pris une place désormais importante dans le monde du logiciel en général, il ne disparaîtra pas demain, et il convient donc de s'intéresser à ses méthodes, aussi originales que son modèle économique.

Chapitre 6 Comment travailler avec des informaticiens ?

Sommaire

Si nous essayons de résumer ce que nous avons écrit au fil des quatre derniers chapitres, 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 ?

6.1 Échapper au développement

6.1.1 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 5.6.3Le développement du logiciel libresubsection.5.6.3, 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.

6.1.2 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'œuvre 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 121) 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'œuvre 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 trente 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 [15].

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.

6.1.3 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.

6.1.4 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.

6.1.5 Le logiciel libre, phénomène technique, social et culturel

Le recours au logiciel libre, dont nous avons déjà abondamment parlé à la sectionn 5.6Le modèle du logiciel libresection.5.6, 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.

6.1.6 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 [108] *(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 œuvre 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

1 Je propose la traduction *appliquette*, ou *applète*.

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*) [44] ; 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*.

6.2 Et s'il faut malgré tout développer ?

6.2.1 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'œuvre : ce point sera développé à la section suivante.
4. Il faut lire le livre de Brooks [24] et un livre sur l'*eXtreme Programming* [9] : 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.

2 Résumé d'une communication de Jean-Marie Faure au Club des Maîtres d'Ouvrage.

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.

6.2.2 Équilibre entre maîtrise d'ouvrage et maîtrise d'œuvre

Parmi les conditions récurrentes d'échec de projets informatiques figurent les dysfonctionnements de la maîtrise d'ouvrage, de la maîtrise d'œuvre, 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'œuvre, campait sur une position de pouvoir inexpugnable, fortifiée par le coût considérable des investissements informatiques. C'était le maître d'œuvre 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'œuvre 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'œuvre 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.

6.2.3 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 compétence[3]. 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

3 Il 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.

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 – 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 $\frac{1}{n} \times (\textit{compétence du chef})$, 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 [6], 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 [23], 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 2.4.4Les détails de la conduite de projetItem.10 et 4.3.2Régie ou forfait : à bon chat, bon ratsection*.27).

6.2.4 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

4 Les 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 cœur de la biologie moléculaire contemporaine[15].

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.

6.2.5 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 œuvre : 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 4.2.6Où gît la difficulté de la programmationsubsection.4.2.6 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.

Chapitre 7 L'horizon du système d'information

Sommaire

7.1 Cohérence *et* pertinence des données

À son apparition, l'informatique de gestion donna naissance à des applications séparées : comptabilité, paie du personnel, gestion des stocks. Il apparut bientôt que la séparation de ces applications engendrait des redondances et des incohérences dans les données. D'où l'idée d'unifier les différentes visions de l'entreprise exprimées par les fichiers multiples de ces applications en un système cohérent : le Système d'Information de l'Entreprise [29] (SI).

Cette idée est bien sûr pleine de bon sens : dès lors qu'une donnée, disons la catégorie d'un employé, est enregistrée de façon indépendante dans le fichier de paie d'un côté, dans l'organigramme de l'entreprise de l'autre, il est à peu près inévitable que ces données soient incohérentes. C'est pour éviter cela qu'ont été inventés les Systèmes de Gestion de Bases de Données (SGBD).

L'idée de Système d'Information est bien sûr une bonne idée, mais ici aussi nous allons voir que sa systématisation outrancière conduit à l'inverse du résultat recherché, à la perte d'information.

Comme nous avons déjà eu l'occasion de le voir, notamment à la section 3.5.1, l'information n'est pas une denrée qui existe dans la nature, elle est le produit d'une élaboration complexe par une démarche que nous pouvons décomposer en deux temps : une phase de construction d'une base de données, suivie d'une phase d'interprétation de ces données. Les données contenues dans la base dépendront étroitement des intentions qui auront été au principe de leur élaboration.

Nous examinerons tout d'abord quelques exemples qui nous aideront à appréhender la difficulté de la conception d'un Système d'Information. Nous vérifierons notamment un résultat fondamental de la *Field theory of information* évoquée à la page 56, selon lequel une base de données construite pour observer et décrire un phénomène ne peut pas être impunément utilisée pour en décrire un autre, même si l'intitulé des données donne à penser qu'elles conviennent aux deux usages. Ce résultat, qui sera illustré par des exemples simples que chacun pourra vérifier, jette un doute sur la validité de projets tels que les *datawarehouses* ou les SI décisionnels construits en puisant directement les données dans les bases destinées à l'exploitation quotidienne de l'entreprise. On pourra trouver une confirmation *a contrario* de ce doute en consultant sur le site de Michel Volle [122] une histoire de *datawarehouse*

qui a réussi ; on verra notamment que le prix de cette réussite a été un effort considérable pour élaborer de nouvelles données à partir des sources.

7.1.1 Inventaire et budget

Dans plusieurs organismes successifs, j'ai eu la responsabilité des systèmes et des réseaux informatiques, et j'ai été confronté à la nécessité d'avoir une idée la plus précise possible du parc de micro-ordinateurs, imprimantes et autres réseaux locaux installés par les utilisateurs, afin d'évaluer correctement la dimension des infrastructures à installer. Le problème semblait minuscule et simple, et les gestionnaires du lieu m'ont conseillé d'explorer pour ce faire le fichier des immobilisations où étaient enregistrés tous les matériels acquis par l'organisme et suffisamment onéreux pour avoir le statut d'investissement.

La méthode ainsi suggérée soulève dès l'abord une objection de fond : un inventaire des immobilisations omet, par définition, tous les matériels loués ou achetés en crédit-bail, que la comptabilité connaît sous la rubrique « Travaux, fournitures et services extérieurs », très éloignée des investissements, de telle sorte que le rapprochement et la mise en cohérence de ces deux sources d'informations est pratiquement impossible. Sans aller plus loin, on voit déjà que le responsable informatique, à la recherche de données relatives aux objets physiques de son domaine, trouvera difficilement ce qu'il cherche dans les informations de gestion.

L'enregistrement des immobilisations est une obligation légale, mais les données ainsi collectées sont rarement exploitées. Des données inexploitées ne stimulent pas la mise au point de bonnes procédures de collecte : partout où j'ai eu l'occasion de les examiner, les données relatives aux immobilisations étaient de très mauvaise qualité. Les montants étaient à peu près exacts, mais les intitulés censés décrire les objets acquis étaient totalement inexploitables, rédigés par des personnes qui n'avaient pas la moindre idée de ce dont il s'agissait ni de comment se rapprocher d'une nomenclature systématique. De toutes les façons, les utilisateurs s'étaient ingéniés à acheter des matériels parfois importants en mettant à contribution les sources de financement les plus variées, dont certaines n'entraînaient aucune inscription dans les livres de l'organisme. Les matériels acquis avec l'argent de contrats de recherche divers et variés n'étaient même pas la propriété de l'entreprise, mais il me fallait néanmoins les raccorder au réseau, et donc les connaître. Bref, l'exploitation des bases de données opérationnelles ne donnait aucun espoir d'obtenir une idée même approximative du parc installé, alors que le nom des données en laissait espérer une description exacte.

Dans un des organismes en question, j'ai procédé par enquête, avec un questionnaire distribué à l'ensemble des responsables de services pour leur demander la liste des matériels installés dans leur secteur. Je venais d'y prendre mes fonctions, ce qui me faisait bénéficier de l'effet de surprise, et comme la construction d'un nouveau réseau était à l'ordre du jour, le bruit s'est répandu que ceux qui ne répondraient pas au questionnaire ne seraient pas raccordés. Ces deux circonstances ont conféré à l'enquête un taux de réponse excellent, qui a permis d'élaborer une estimation du parc installé dont l'expérience ultérieure a montré qu'elle n'était pas trop mauvaise. On peut noter que les informations recueillies, parfaitement adaptées à l'objectif poursuivi, ne pouvaient être d'aucune utilité aux comptables chargés de gérer les immobilisations. Par la suite, des enquêtes analogues ont été lancées pour mettre à jour les données de la première, mais le stratagème avait été éventé, et les taux de réponse ont été si faibles que les résultats étaient à peu près inexploitables. Je vérifiais ainsi une règle de base du métier de collecteur de données : l'information est une denrée qui s'échange, les gens que l'on interroge ne vous répondent utilement que s'ils y voient un intérêt, et un ordre de répondre émis par la hiérarchie ne fait que diminuer la qualité des réponses.

Je retiendrai de ces expériences convergentes qu'essayer d'utiliser pour les besoins de la gestion technique une base de données constituée pour les besoins de la conformité aux

règles comptables était voué à un échec total malgré l'identité nominale des données utiles aux deux usages.

Une autre petite expérience que chacun a pu faire : je devais gérer le budget de mon service et donc veiller à ne pas émettre de commandes qui en excèdent le solde. En principe le service comptable de l'entreprise est là pour résoudre ce genre de problème, mais entre l'instant où une commande est émise, celui où la somme correspondante est débitée, et plus encore celui où cette information parvient à l'acheteur, court un délai qui peut être long, phénomène dit de la différence entre comptabilité de trésorerie et comptabilité au fait générateur. En principe la commande est enregistrée au stade de l'émission pour les besoins de la gestion de trésorerie, mais cette information n'est pas forcément accessible au service acheteur (inutile de dire que dans les administrations publiques on n'a même pas conscience de l'existence de la question). Les systèmes de commande en ligne résolvent en partie ce problème, mais jusqu'à une date récente chacun devait tenir sa petite comptabilité parallèle pour éviter les dépassements fâcheux.

7.1.2 Enquête statistique, SI géographique

L'INSEE organise des *Enquêtes Annuelles d'Entreprises* (EAE) qui sont une des bases de l'étude de l'activité économique en France. La nature de l'enquête est adaptée aux différents secteurs de l'industrie et des services. J'ai eu la chance il y a quelques années de participer à la refonte de l'EAE Bâtiment - Travaux Publics, une expérience extrêmement instructive[1]. Incidemment, le maître d'œuvre de cette EAE particulière était à l'époque la Division Statistique du Ministère de l'Équipement.

Il fallait élaborer un questionnaire qui satisfasse à trois conditions :

— il devait procurer les informations attendues par les organisateurs de l'enquête ;
— la réponse aux questions devait ne pas poser de problèmes insolubles aux entreprises interrogées ;
— l'organisation générale de l'enquête devait se faire en collaboration avec les deux syndicats professionnels, la Fédération Nationale du Bâtiment et celle des Travaux Publics.

Le Bâtiment et les Travaux Publics sont des activités assez différentes. Il y avait à cette époque 230 000 entreprises de bâtiment, dont 210 000 entreprises d'une seule personne, cependant que les 15 000 entreprises de travaux publics étaient en moyenne beaucoup plus importantes.

Demander à une très petite entreprise de remplir un questionnaire statistique est délicat : il ne faut poser qu'un minimum de questions, pour lesquelles l'élaboration de la réponse ne demande pas des heures de travail, c'est-à-dire que les chiffres à fournir doivent figurer tels quels dans les livres comptables de l'entreprise interrogée. Nous voulions connaître les effectifs, la masse salariale, le chiffre d'affaires, choses assez simples, mais aussi les investissements, ce qui est une notion déjà beaucoup plus complexe, notamment du fait des amortissements et des éventuels dégrèvements fiscaux ou subventions qui l'accompagnent. Il fallait aussi des renseignements qualitatifs sur les gros matériels. Cela m'a donné l'occasion d'une lecture attentive du plan comptable de la branche, et d'une augmentation significative de ma compétence (livresque) en matière de bulldozers et de sonnettes à vapeur.

Mais la principale surprise me fut procurée par les négociations avec les syndicats professionnels. Nous avons eu l'honneur d'être reçus par le Président de la Fédération Nationale du Bâtiment : en fait ce monsieur assez important ne demandait pas mieux que de donner sa bénédiction à notre enquête, il suffisait de lui demander – et de faire passer l'administration du questionnaire par ses services, ce qui avait l'avantage pour nous qu'il parviendrait aux entreprises interrogées sous le sceau de leur syndicat, et pour le syndicat de lui donner la

1 Merci à Jean-Michel Agnus, qui dirigeait les opérations !

primeur des résultats bruts. Finalement tout s'est déroulé à la satisfaction générale, mais il est clair que l'omission d'une seule de ces étapes diplomatiques aurait suffi à liquider radicalement la qualité de l'enquête pour plusieurs années.

De cette expérience de construction d'un système d'information – en effet qu'est-ce qu'une enquête statistique, sinon un SI ? – nous avons retenu que pour obtenir l'accès aux données de base, et pour en extraire de quoi fabriquer l'information que nous voulions, il nous a fallu d'une part nous livrer à une étude très spécifique de l'univers particulier que nous nous proposions d'explorer, d'autre part mener des négociations somme toute agréables bien qu'assez serrées avec les membres de cet univers.

Lorsque les instituts de statistique publics traversent des saisons d'étiage budgétaire, ils n'ont plus les moyens d'organiser toutes les enquêtes qu'ils souhaiteraient, et ils se rabattent sur l'exploitation des données administratives. Parfois cela donne des résultats exploitables, mais si cette pratique est peu prisée c'est bien parce que la plupart du temps les données extraites des bases administratives ne satisfont pas aux impératifs de la statistique.

Prenons un dernier exemple dans le domaine des systèmes d'informations géographiques. La question fut posée un jour d'une possible synergie entre le service du cadastre, qui fait des cartes à grande échelle à des fins surtout fiscales et notariales, et l'Institut Géographique National, qui fait des cartes topographiques à grande échelle pour les militaires et les randonneurs ; la réponse fut évidente, mais imprévisible pour le profane : le cadastre est confronté à une exigence de conservation des surfaces, cependant que pour l'IGN c'est la conservation des distances qui est importante, de ce fait le cadastre et l'IGN utilisent des méthodes cartographiques fondées sur des projections différentes et parfaitement incompatibles. Les données de l'un ne seraient d'aucun secours pour l'autre.

7.1.3 Développement de logiciel statistique

Revenons sur le cas du logiciel statistique dont nous avons évoqué le développement à la page 42. Pourquoi les experts de l'INSEE, assistés par d'excellents ingénieurs d'une société de services réputée, n'ont-ils pas réussi à construire un logiciel aussi abouti que SAS (*Statistical Analysis System*), par exemple ?

SAS, de SAS Institute [99], est aujourd'hui et depuis près de trente ans le logiciel préféré des statisticiens, qui ne trouvent guère à lui reprocher que son coût élevé. Vu d'un œil d'informaticien il y aurait bien des choses à lui reprocher, notamment la lourdeur de sa syntaxe et de sa gestion de données, son caractère fermé, etc. – rien n'y fait, les statisticiens l'adorent. On observe ici un phénomène assez courant dans les usages de l'informatique : imposer à l'utilisateur un apprentissage long et pénible peut conduire le logiciel à l'échec, mais si cet écueil a été évité et si pour quelque autre concours de facteurs positifs le système a réussi à convaincre une masse critique d'utilisateurs enthousiastes, la difficulté même de l'apprentissage renforce l'adhésion de ceux qui l'ont subi, qui semblent ne pas vouloir abandonner les fruits d'une compétence si chèrement acquise. Il entre aussi sans doute dans cet adhésion une part de fierté à être admis dans une société d'initiés. SAS a de toute évidence réussi à déclencher ce phénomène, dont un autre exemple est le langage de programmation C, à la syntaxe particulièrement torturée.

SAS a été à ses débuts l'œuvre de quelques enseignants et chercheurs de l'université d'État de Caroline du Nord, au nombre desquels l'actuel PDG de SAS Institute James Goodnight, qui avaient conçu ce logiciel pour leurs propres besoins. La société fut créée en 1976 et établit presque aussitôt un partenariat avec IBM. Il convient de souligner que le marché des logiciels statistiques n'était pas vierge, mais l'intégration de SAS aux systèmes IBM et son interface interactive ont contribué à un succès rapide. Le système se présente comme un interpréteur de commandes : l'utilisateur tape une expression du langage d'interrogation, le système donne la réponse. Ce n'est pas forcément idéal pour des travaux complexes avec

de gros volumes de données, mais l'effet de séduction initiale est garanti – et en 1976 c'était inhabituel.

Parmi les éléments qui ont assuré le succès continu de SAS depuis 1976, nous citerons la qualité statistique du système : chaque fois que j'ai eu l'occasion de le vérifier moi-même ou de recueillir l'avis de spécialistes de domaines particuliers des statistiques, j'ai constaté que SAS offrait la gamme complète des meilleurs algorithmes programmés de façon impeccable. Cette qualité reconnue donne au logiciel un statut envié auprès des revues scientifiques, dont les *referees* acceptent sans vérification les résultats des calculs dès lors que ceux-ci ont été effectués avec SAS. Autre qualité du logiciel, l'utilisateur peut étendre ses possibilités en lui ajoutant des fonctions qu'il programme lui-même, ce qui est de première importance pour des chercheurs, qui par définition ne sauraient se satisfaire des analyses standard. SAS Institute sait maintenir avec ses clients un contact étroit et attentif, notamment au travers de clubs d'utilisateurs très dynamiques, ce qui lui a permis de s'adapter parfaitement aux évolutions du marché – ainsi on chercherait en vain sur la page d'accueil de leur site WWW le mot « statistique », et c'est plutôt l'analyse financière qui y est en vedette. Enfin SAS Institute n'est pas une société par actions, ce qui lui permet d'échapper aux exigences d'actionnaires et de maintenir un budget de recherche et de développement particulièrement élevé, plus de 25% d'un chiffre d'affaires qui s'élevait en 2003 à 1,34 milliard de dollars.

Revenons à la question posée au début de cette section : pourquoi les développements de l'INSEE ont-ils abouti à un succès moindre que celui de SAS ? Si l'on compare les moyens mis en œuvre, il ne fait aucun doute que les effectifs d'experts en statistiques et d'ingénieurs en informatique de l'INSEE étaient supérieurs à ceux de l'université d'État de Caroline du Nord, et l'équipe du projet logiciel de l'INSEE comptait dix personnes alors que SAS Institute avait sept employés en 1976.

Il est sans doute injuste de formuler la question en ces termes : après tout, dans le domaine du logiciel comme dans bien d'autres, toutes les tentatives ne peuvent pas aboutir au même succès, et l'équipe de l'université d'état de Caroline du Nord comportait sans doute des personnages particulièrement compétents, dotés d'une vision aiguë de l'avenir à moyen terme des usages de l'informatique et des moyens techniques de les satisfaire. L'aide d'IBM pourrait bien avoir été décisive. Observons cependant que SAS a été développé initialement par des experts pour leur propre usage, tandis que le logiciel de l'INSEE a fait l'objet d'un cahier des charges rédigé par une équipe non dépourvue de compétences, mais orientée vers l'obtention de gains de productivité par des méthodes tayloriennes.

D'autre part, le cahier des charges était très conditionné par les pratiques en cours à l'INSEE pour le dépouillement d'enquêtes statistiques, qui étaient soumises à des contraintes particulières. L'enquête par excellence, qui constituait et constitue toujours la mission centrale de l'INSEE, c'était le recensement de la population, qui comportait (cela va changer) un enregistrement par habitant du pays et un enregistrement par logement, soit 90 millions d'enregistrements. À l'époque dont nous parlons il était inimaginable de placer cet énorme volume de données sur disque, et le traitement était effectué à partir de fichiers sur bandes magnétiques. Les stipulations techniques du cahier des charges étaient donc très orientées vers des traitements par lots de fichiers séquentiels, méthode de travail qui était imposée par les circonstances mais qui ne répondait pas à tous les besoins. Le logiciel était assez bien adapté à l'obtention de tableaux de base à partir de données brutes, mais les chercheurs de l'INSEE qui devaient ensuite réaliser des analyses plus fines ne disposaient ni de méthodes modernes et appropriées de gestion des données, ni de méthodes statistiques perfectionnées, ni de moyens d'étendre le logiciel par l'ajout de fonctions à leur convenance. Ils eurent donc tendance à utiliser plutôt SAS.

7.1.4 Passage à l'Euro

Mon employeur me confia un beau jour de juillet 2001 le pilotage du passage à l'Euro de ses applications financières et comptables. J'étais depuis deux mois dans la maison, la transition devait être achevée pour le 1er janvier 2002, sans report possible puisque c'était la date limite fixée par la loi, et le chef du projet Euro venait de donner sa démission. Je dois dire que cette situation était un peu inquiétante.

Le projet Euro avait été mené selon toutes les règles de l'art, et donc un serveur de fichiers consacré au projet abritait des dizaines de documents, représentant au total quelques milliers de page. Je plongeai fébrilement dans cette masse de documentation : la récolte que je pus en tirer fut conforme aux règles de l'art des projets, c'est-à-dire que ces documents formels et verbeux étaient conformes aux normes mais d'une vacuité inquiétante en termes d'information réelle, ils ne contenaient à peu près aucune indication ni sur la situation de départ, ni sur ce qu'il convenait de faire. J'y trouvai quand même quelques généralités relatives aux règles comptables de passage à l'Euro, qui tenaient en une dizaine de lignes soigneusement recopiées sur des dizaines de documents, et une liste à peu près complète des applications concernées.

J'étais épaulé par un prestataire extérieur, chargé d'assistance à la maîtrise d'ouvrage, et par un planificateur, dont les avis convergèrent : il fallait recenser toutes les applications potentiellement concernées par le passage du franc à l'Euro, dresser un tableau où elles figureraient en ligne et en colonne, chaque case correspondrait ainsi à une interférence possible entre deux applications; on pourrait faire abstraction des cases de la diagonale, et ne considérer qu'un des triangles déterminés par cette diagonale en admettant que les interactions entre l'application A et l'application B étaient les mêmes que celles entre B et A. Nous recensâmes vingt applications, ce qui donnait un tableau de 400 cases, ramené par les simplifications évoquées ci-dessus à 180 cases utiles. Examiner ces 180 cas et résoudre les problèmes soulevés avant le 1er janvier 2002, même en y passant les nuits et les week-ends, semblait une mission impossible. Évidemment, cela aurait pu aussi représenter un contrat attrayant pour un prestataire de services...

Cet objectif n'était hors d'atteinte que parce que la méthode proposée par le chargé d'assistance à la maîtrise d'ouvrage et par le planificateur était adaptée à un ordinateur, mais pas du tout à des humains. De toute façon la démarche proposée était non seulement irréalisable, mais en outre peu judicieuse; il fallait faire autrement.

Quelques entretiens avec le directeur du système d'information, fin connaisseur des circuits de données et des habitudes informelles de la maison, m'apprirent quelles étaient les applications vitales, ainsi que les dates réelles auxquelles des résultats seraient exigibles. Ainsi telle application était en fait trimestrielle, ce qui laissait jusqu'au 15 mars pour régler son cas; telle autre ne concernait qu'un fichier de quelques dizaines d'entrées, en cas de malheur on pourrait toujours s'en sortir avec un tableur. Par contre il fallait bien entendu traiter sans faute la paie, le budget, les commandes, puis ne pas trop tarder pour le règlement des fournisseurs.

Cette démarche empirique mais informée à la meilleure source réduisait le nombre d'applications concernées à cinq, mais surtout les interférences réelles à quatre ou cinq, de surcroît très simplifiées dès qu'elles avaient été expliquées par le directeur du système d'information, qui connaissait leur nature, ce qui évitait d'avoir à explorer toutes sortes de cas qui ne se produiraient jamais.

Une fois les problèmes identifiés, je pris contact avec les auteurs ou les responsables de maintenance des programmes concernés, qui par chance étaient encore présents dans la maison, et qui se montrèrent compétents, coopératifs et efficaces. Je dois dire notamment qu'ils connaissaient bien mieux les aspects règlementaires du passage à l'Euro que les gens dont c'était théoriquement la fonction, et qui furent dans cette affaire d'une tranquillité impressionnante. Le passage du Franc à l'Euro posait notamment une question incontour-

nable, la nécessité de choisir entre trois méthodes possibles de conversion à l'Euro des bases de données et des applications :

1. — Première méthode : conversion uniquement au niveau le plus fin. On remonte dans la base de données en respectant les règles de celle-ci. En d'autres termes, on convertit les lignes détail des écritures et on refait les calculs pour obtenir la nouvelle valeur des lignes total.
   — Avantages : Respecte la logique de la base de données.
   — Inconvénients : Les écarts de conversion même s'ils sont faibles peuvent engendrer des incompréhensions pour les utilisateurs (non respect de la cohérence des montants en francs et en euros).
2. — Deuxième méthode : conversion directe de tous les montants de la base de données (lignes détail comme total).
   — Avantages : respecte la cohérence entre les montants en francs et en euros.
   — Inconvénients : ne respecte pas la logique de la base de données. La nouvelle valeur d'une ligne total ne sera pas toujours égale à la somme des valeurs des lignes détail.
3. — Troisième méthode : conversion directe de tous les montants de la base de données. Création d'une ligne de détail supplémentaire pour l'écart de conversion.
   — Avantages : respecte la cohérence entre les montants en francs et en euros et la logique de la base de données.
   — Inconvénients : suivant les caractéristiques de l'application peut être très lourde à mettre en place voire impossible.

Nous attendîmes en vain que les responsables officiels nous indiquent quelle solution choisir. Finalement nous (les informaticiens) eûmes à décider nous-mêmes, ce qui ne semble pas une procédure normale.

Bref tout se passa sans encombre, parce que la méthode canonique fut évitée, et grâce au savoir concret de gens qui connaissaient bien le terrain et les programmes. Le modèle aujourd'hui standard du chef de projet parfaitement incompétent mais qui suit des procédures en béton, illustré par le chef du projet Euro qui avait prudemment démissionné au moment fatidique, aurait mené droit à la catastrophe. Cette expérience plaide aussi pour l'acquisition et la conservation de compétences internes.

7.1.5 Annuaire électronique

J'ai eu un jour à lancer et à réaliser un projet d'annuaire électronique dans un grand organisme de recherche scientifique. Il apparut assez vite que ce type de projet était notablement plus complexe qu'il n'y paraît lorsque l'on n'a pas essayé.

Constituer un annuaire papier, donc statique, d'une population donnée consiste à effectuer à un instant donné un recensement exhaustif des individus de cette population et de leurs caractéristiques qui doivent figurer dans l'annuaire, telles qu'adresse, numéro de téléphone, etc.

Pour constituer un annuaire électronique qui présente des avantages significatifs par rapport à un annuaire papier, il faut créer une base de données qui contienne les données déjà évoquées ci-dessus, et surtout mettre en place des processus d'alimentation et de mise à jour de cette base avec pour objectifs les qualités suivantes : pertinence, actualité, fiabilité, exhaustivité, disponibilité. Concevoir ces processus d'alimentation de l'annuaire demande d'avoir identifié les sources adéquates de données.

La réponse naïve à cette question, qui me fut donnée par quelques aspirants-prestataires, va de soi : « Eh bien, vous prenez votre fichier de personnel, une extraction, et voilà ! » C'était simplement oublier qu'un organisme de recherche réunit bien d'autres individus que ses propres personnels, lesquels ne représentent qu'une petite moitié des quelques milliers de

personnes actives dans l'institution. Pour écarter d'autres idées simplistes, qu'il suffise de dire que lesdits personnels sont dispersés sur quelques dizaines de sites dotés chacun de ses propres règles de gestion administrative et technique. Il fallait chercher autre chose.

Une enquête auprès de collègues expérimentés m'apprit qu'il existait environ 130 fichiers de personnel dans la maison, exhaustifs ou partiels. De quoi faire dresser les cheveux sur la tête d'un concepteur de système d'information ! Il est plus que probable que l'existence de bases de données de bonne qualité facilement accessibles serait de nature à faire disparaître beaucoup de ces fichiers d'intérêt local, constitués pour résoudre un problème ponctuel et pas forcément toujours de très bonne qualité. Mais il ne ferait pas passer le nombre de fichiers de personnel de 130 à 1 !

Une visite aux détenteurs de quelques-uns des 130 fichiers a révélé un certain nombre de fichiers redondants, morts ou indigents, mais aussi des fichiers bien vivants qui avaient de bonnes raisons de continuer à mener une existence distincte. Parmi les bonnes raisons, la plupart ont trait à des exigences temporelles quant à la disponibilité des données. Ainsi, lorsqu'un nouveau salarié prend ses fonctions le premier jour du mois, le degré d'urgence de son inscription dans les fichiers du personnel est déterminé par l'objectif de lui verser sa rémunération, c'est-à-dire que cette inscription doit avoir lieu au plus tard entre le 15 et le 20 du mois. Mais pour qu'il puisse effectivement commencer à travailler il faut lui ouvrir un compte de messagerie électronique bien avant cette date, et, pour ce faire, l'enregistrer dans les bases de données correspondantes. Il serait également souhaitable qu'il soit dans l'annuaire téléphonique. Bref, les auteurs et les utilisateurs de ces fichiers ont des impératifs différents, sans même aborder la délicate question de la confidentialité des données et du secret professionnel. Les supprimer au profit d'une base de données unique n'irait sûrement pas sans poser de difficiles problèmes.

Il existe dans cet organisme une base d'informations sur les recherches, qui contient des entrées pour chaque formation de recherche (unité, laboratoire, équipe, etc.) et pour chaque chercheur, ingénieur ou technicien. Chaque entrée comporte des informations factuelles (adresse, nom de l'entité, liste du personnel, liste de publications…) ou descriptives (résumé des recherches en cours). Lorsque les formations de recherche rédigent leur demande budgétaire annuelle, celle-ci n'est validée que si la base de données est mise à jour ; de plus, le budget attribué croît avec les effectifs enregistrés dans la base, ce qui incite à ne pas oublier de personnels. Ce dispositif bureaucratique garantit une bonne mise à jour une fois par an, ce qui n'est déjà pas si mal. Il est question de modifier cette règle, justement parce qu'elle est bureaucratique (ô combien : les unités de recherche doivent remplir chaque année 22 formulaires pour mettre à jour cette base !), ce qui sera une bonne idée mais ne fera pas les affaires de notre projet d'annuaire. Il est clair que, lorsque la mise à jour de la base ne sera plus un événement déclencheur de financement, les formations de recherche seront moins enclines à veiller à son exactitude (on les comprend).

Cela dit, la base d'informations sur les recherches nous semblait la meilleure source de données pour notre annuaire, malgré plusieurs faiblesses :

— elle n'était mise à jour qu'une fois par an ;
— son développement avait été confié à un prestataire extérieur dont l'activité avait été peu contrôlée, de ce fait le Modèle Conceptuel de Données de la base avait dérivé vers un état peu connaissable ;
— le schéma de la base s'était dégradé, sa structure opacifiée ;
— les identifiants de certaines tables étaient de mauvaise qualité ;
— la base avait été maintenue dans un état relativement bon par la personne qui l'avait conçue et créée, mais cette personne était partie à la retraite, et dans ce genre de situation la succession est toujours, disons, difficile.

Que le lecteur ne se laisse pas abuser par l'apparence anecdotique de cette liste de défauts : toute base de données est bien réellement exposée à ces risques précisément.

Nos dernières illusions s'envolèrent lorsque nous décidâmes de quitter le havre de la Direction Générale pour visiter des sites opérationnels en province ou en région parisienne. Les personnes chargées d'alimenter les bases en données recueillies sur le terrain nous expliquèrent avec ménagement mais franchise les procédures suivies. Les fichiers dont le contenu avait une incidence financière ou en termes de personnel étaient mis à jour sérieusement, mais uniquement pour les données qui avaient une telle incidence. D'autres fichiers à usage purement bureaucratique (ou perçu comme tel), et dont les procédures de mise à jour étaient de surcroît particulièrement pénibles, étaient beaucoup moins bien traités – en général on se contentait de renvoyer la version de l'année précédente et personne ne s'apercevait de rien.

Bref, nous avions rêvé d'un système d'information où il nous aurait suffi de voleter de base de données en base de données pour y butiner les données utiles à notre projet : il se révélait que nous avions à construire les données dont nous avions besoin, et que la réutilisation de données existantes n'était pas un avantage mais bien plutôt une contrainte, imposée par le souci de cohérence mais assortie d'un coût élevé induit par leur mauvaise qualité et, paradoxalement, par leur incohérence, qu'un avantage.

Je ne crois pas que la situation décrite ci-dessus corresponde à un cas particulièrement défavorable : je pense au contraire que tous les univers de données réels sont peu ou prou conformes à cette description. Les données sont bonnes si elles ont une bonne raison de l'être, et elles sont bonnes à l'usage pour lequel elles ont été construites. Si on veut en faire autre chose, il faut les ré-élaborer entièrement. On pourra se reporter utilement à l'« Histoire d'un tableau de bord » narrée par Michel Volle [123] ; à partir de données de gestion il s'agit de créer un tableau de bord pour la direction d'une grande entreprise ; ce ne sera rien de pharaonique, juste une sélection de séries temporelles soumises à des traitements statistiques classiques de correction des variations saisonnières et d'extrapolation de tendance ; l'objectif a été atteint, mais les données n'étaient pas juste là, prêtes à servir, comme le lecteur pourra s'en rendre compte en lisant l'article. Dans le même ordre d'idées on pourra aussi consulter un article consacré à la réalisation, dans le même contexte, d'un *datawarehouse* [122], c'est-à-dire d'un système informatique d'aide à la décision dans le domaine commercial à partir des données opérationnelles : le système d'extraction et d'adaptation des données est une véritable usine informatique qui a coûté 25 millions d'Euros.

7.1.6 Informatique de gestion manuelle

Au tout début des années 1990, j'ai été amené à contribuer au renouvellement de l'informatique de gestion d'un grand établissement d'enseignement supérieur. Il s'agissait d'organiser le renouvellement complet du système, matériel et logiciel. À cette occasion je me suis penché sur le fonctionnement informatique du système de paie, qui s'est révélé assez curieux : il existait un logiciel de paie, ainsi qu'un fichier des personnels devant recevoir une rémunération. Chaque mois ce programme était lancé, et il produisait les quelques centaines de bulletins de salaire qui correspondaient aux personnes qui avaient travaillé ce mois-là. Mais ces bulletins étaient pour la plupart lourdement erronés. En fait ils ne servaient que de masques de saisie pour une dame du service de paie qui possédait la mémoire et les arcanes de la paie de chaque agent. Elle corrigeait tous les bulletins à la main, et lançait l'impression des vrais bulletins de salaire. Personne n'avait jamais réussi à paramétrer correctement le logiciel, et le traitement était en fait manuel. Cette dame prenait ses vacances en août, mois pour lequel on rejouait la paie de juillet, quitte à régulariser en septembre les différences qui auraient dû apparaître entre juillet et août.

Ce secret était bien caché, et sa découverte m'a valu l'inimitié de la dame. L'avantage d'une paie manuelle, c'est qu'elle est facile à personnaliser : mon salaire du mois suivant fut de 45 francs et quelques centimes. Mais je suis convaincu que cette pratique (la paie manuelle, pas le salaire de 45 francs) est plus répandue qu'on ne le pense. Lorsque, pour tel

système de gestion financière et comptable, des prestataires font intervenir des dizaines de consultants pendant des mois pour des « réfections de bases de données », de quoi s'agit-il en réalité, sinon de passer des écritures comptables à la main ? Cela ne révèle-t-il pas un dysfonctionnement grave du système ?

7.2 Terminologie, thésaurus, « ontologie »

L'enthousiasme engendré par une percée intellectuelle telle que le modèle objet (par exemple) peut susciter une tendance à en surestimer la portée, à en prendre les auteurs pour des démiurges[2]. La propension à verser dans cette erreur résulte souvent d'une mauvaise perception de la complexité des rapports entre les mots inventés par les humains et les choses dans la réalité : des naïfs peuvent croire que ces rapports seraient en fait une relation isomorphe, c'est-à-dire qu'à chaque mot correspondrait une chose dans le monde réel, et que la structure des relations entre les mots refléterait simplement celle des relations entre les choses réelles. Nous allons très brièvement et très sommairement expliquer ici pourquoi le point de vue qui considère le monde des significations comme un simple reflet du monde réel nous semble faux, puis envisager quelques occurrences de cette erreur dans le monde des Systèmes d'Information. Nous renvoyons le lecteur qui voudrait prendre connaissance d'une véritable analyse philosophique de la question à la lecture, par exemple, d'un livre de Karl Popper [91] qui lui consacre des développements détaillés et approfondis. Roger Penrose [89] donne aussi des arguments contre la thèse de l'isomorphisme entre les mots et les choses. Nous avons d'ailleurs déjà rencontré à la section 3.5.1*Total Data Quality Management (TDQM)*subsection.3.5.1 un point de vue de ce genre (la « théorie TDQM »), et sa réfutation par Isabelle Boydens [21].

Pour parler du monde qui nous entoure, nous employons un langage fait de signes, et nous avons appris à interpréter ces signes de telle sorte que nous pouvons grâce à eux échanger avec nos congénères quelques informations qui éventuellement pourront nous renseigner (ou nous induire en erreur) sur certains aspects de certains objets du monde réel. Des propos de notre interlocuteur, inférer ce qu'il a voulu nous dire, ce qu'il « avait derrière la tête » et qu'il voulait nous communiquer, est déjà un exercice où le risque d'erreur (de malentendu) est important ; pour comprendre ces propos, nous devons posséder toutes sortes de compétences relatives au contexte dans lequel a lieu l'échange de signes et à la nature particulière de cet interlocuteur et de nos relations avec lui. Alors *a fortiori* entre les signes du langage qu'il aura employés et un objet, ou un état, ou un événement du monde réel, la distance laissée à l'interprétation est considérable, et c'est là tout le problème de la *représentation*, qui n'a pas une réputation de facilité. Croire dans ces conditions qu'un langage humain (tel que le français ou le bambara, au hasard) puisse rendre compte « exactement et naturellement » ne serait-ce que d'une partie infime de l'univers, ce n'est pas seulement nourrir une illusion, c'est se méprendre sur la nature même du langage, du monde et de leurs rapports. N'oublions pas que « le langage travestit la pensée. Et notamment de telle sorte que d'après la forme extérieure du vêtement l'on ne peut conclure à la forme de la pensée travestie ; pour la raison que la forme extérieure du vêtement vise à tout autre chose qu'à permettre de reconnaître la forme du corps [132] ».

Victor Klemperer, philologue allemand qui a miraculeusement échappé au génocide, a consacré à la langue nazie un livre [62] où il écrit (page 35 de l'édition française *Pocket*) : « On cite toujours cette phrase de Talleyrand, selon laquelle la langue serait là pour dissi-

2 La tentation de se croire égal aux dieux, d'excéder ses limites, porte le nom savant d'*hybris* ; nous avions eu l'occasion à la section 4.2.5Excès dans la penséesection*.24 d'en signaler une occurrence, qui consistait à imaginer que justement le modèle objet pouvait représenter naturellement le monde réel. Nous avions dit que cette hypothèse n'avait tout simplement aucun sens, parce que rien ne peut « représenter naturellement le monde réel ».

muler les pensées du diplomate… Mais c'est exactement le contraire qui est vrai. Ce que quelqu'un veut délibérément dissimuler, aux autres ou à soi-même, et aussi ce qu'il porte en lui inconsciemment, la langue le met au jour ». Ce passage, loin de contredire la citation précédente de Wittgenstein, la complète et en achève le sens, qui réfute le réductionnisme linguistique : comment, si le langage travestit le propos du locuteur et si sa langue révèle ce qu'il dissimule, peut-on espérer établir une correspondance isomorphe entre discours et réalité ?

Outre les difficultés liées à la labilité des langues humaines et à leur interprétation, il en est une autre qui s'oppose absolument au projet de décrire de façon exacte et complète une partie de la réalité, et William Kent [61] a attiré notre attention sur elle : « Il nous est impossible de tracer un cercle imaginaire autour d'un certain corpus d'information et de déclarer qu'il contient tout ce que nous savons au sujet d'une certaine chose, et que tout ce qui est contenu dans le cercle a trait uniquement à cette chose, et donc que cette information "représente" la chose. »

Les inepties autour de l'aptitude supposée du modèle objet à mettre en correspondance les objets du monde, ceux de la pensée et ceux du langage ne sont que cuistrerie vénielle et hybris modérée à côté des exactions de la branche de la psychologie cognitive qui élabore de soi-disant *ontologies*.

De longue date les spécialistes de la documentation élaborent des thésaurus, qui sont des vocabulaires contrôlés, normalisés et commentés. Leur utilité est grande pour indexer articles et documents : si chaque auteur donne pour son article les mots-clés qui lui passent par la tête avec des acceptions de son cru, les recherches documentaires risquent de devenir difficiles et peu fructueuses. Disposer d'un thésaurus qui explique quel mot utiliser dans quelle circonstance, et qui en donne les équivalents dans différentes langues, est éminemment précieux, même si le rêve d'un thésaurus complet et parfait est inaccessible, ne serait-ce qu'à cause des évolutions incessantes tant des sciences que des langues.

La démographie dispose ainsi du thésaurus POPIN, la biologie de *MeSH*. Tout ce que nous avons dit nous incite à la prudence à leur égard : de tels thésaurus ne sont utilisables que dans le cadre limité pour lequel ils ont été conçus, c'est-à-dire indexer des articles scientifiques. Et un thésaurus multilingue ne saurait prétendre abolir toutes les différences subtiles qui existent entre les langues et remplacer les dictionnaires linguistiques qui, pour chaque entrée, peuvent donner de nombreuses traductions. Il se trouve par exemple qu'en français le mot *grue* peut désigner un oiseau ou un appareil de levage. Le mot anglais *crane* possède les deux mêmes acceptions, ainsi d'ailleurs que son équivalent allemand *Kran*. Mais en français le mot *chemise* peut désigner soit un vêtement, soit une pièce de moteur à pistons : or ni l'anglais ni l'allemand ne possèdent ce couple d'acceptions pour un même mot. Bref, selon que notre thésaurus sera ornithologique, mécanique, vestimentaire ou de travaux publics, il n'aura pas la même structure. Et encore ne s'agit-il ici que d'exemples simplissimes pris dans des langues très voisines.

Les travaux issus de cette entreprise terminologique laborieuse et méconnue mais utile à la science ont été détournés de leur objectif par des cognitivistes qui se sont dit un peu vite que, puisque l'on avait des mots, cela ferait bien l'affaire pour représenter des concepts, et que dès lors que l'on disposait de concepts, pourquoi ne pas les relier par des flèches qui matérialiseraient, par exemple, les formules de la logique du premier ordre, avec ses prédicats qui permettent le calcul logique. Les flèches seraient bien sûr instanciées dans des bases de données informatiques par des pointeurs ou des références, et l'on pourrait ainsi donner une structure formelle de la science et distinguer les énoncés vrais des faux – rien de moins. Et tant qu'à faire, on appellerait cela une ontologie, ou plutôt *des* ontologies, une par domaine scientifique, par exemple.

Il y a là une confusion tout à fait dommageable entre les objets de la linguistique, ceux de la logique et ceux de la métaphysique. Les logiciens sérieux ont compris depuis long-temps que leur discipline devait se doter de ses propres formalismes justement parce que

les langages humains étaient bien trop mobiles et fluides pour des opérations formelles. Comment formuler des propositions sûres avec les mots et les constructions grammaticales du langage humain, dès lors (pour ne prendre qu'une objection entre cent) que telle langue va exprimer telle signification par un procédé lexical, alors que telle autre le fera par un procédé syntaxique ? Ainsi, le français dira « j'ai oublié ma valise à la maison » là où le bambara dira « ma valise est restée à la maison », et d'ailleurs le verbe bambara qui signifie « oublier » n'admet que des compléments d'objet indirects, il n'existe aucun moyen direct d'oublier *quelque-chose*. Les ambiguïtés du langage humain ne sont d'ailleurs pas un simple problème de syntaxe, ce sont les limites mêmes au traitement de l'information par l'esprit humain. Dès la fin du XIX^e^ siècle Gottlob Frege [42] avait compris les difficultés liées à l'usage du langage humain et avait entrepris d'élaborer un système formel pour les démonstrations logiques. Il suivait d'ailleurs en cela un chemin déjà ouvert par George Boole (1815 – 1864).

Certains cognitivistes contemporains semblent ne jamais avoir entendu parler de ces travaux, ou alors ne pas en avoir compris la portée. Il en résulte des confusions dramatiques, exprimées par exemple ainsi [52] : « les terminologies ne peuvent plus se contenter de recenser les termes et les organiser [*sic*] brièvement sous forme d'une hiérarchie. Elles doivent également proposer toute une gamme de relations qui reflètent au mieux les connaissances du domaine et répondent de manière adaptée aux besoins des applications. » On peut lire aussi, plus loin dans le même article qui fait une recension de travaux marquants de ce domaine, à propos d'un système formel à usage médical construit autour de la terminologie SNOMED : « Les règles du système formel utilisent et explicitent les relations, transversales ou non, qui peuvent exister entre les composants du nouveau terme [construit par le système, nda] mais aussi entre le nouveau terme et ses composants. Une règle de formation d'un diagnostic par combinaison d'un terme signifiant une région du corps (axe Topologie de la SNOMED) et d'un terme signifiant une pathologie (axe Morphologie) est par exemple instanciée dans la combinaison de termes élémentaires *glande apocrine* et *inflammation* (pathologie). La nouvelle notion *inflammation de la glande apocrine*, désignée également par le terme *hidrosadénite*, `est-une` *inflammation* qui est `localisé-dans` la *glande apocrine*. Cette règle très productive est également à l'origine d'autres diagnostics… Le potentiel combinatoire des 5 880 termes de l'axe Morphologie de la SNOMED avec les 12 936 termes de l'axe Topographie, permettrait de créer 76 millions de concepts… » On croit rêver – et on espère ne jamais tomber entre les mains de médecins convertis à des systèmes formels de cet acabit. On remarque aussi que les opérateurs du système, écrits dans une police de caractères particulière, comme `localisé-dans`, en reçoivent une aura informatique spéciale qui semble autoriser les fautes d'orthographe.

François Rastier [95] a donné une critique salubre de cette insondable cuistrerie, en passant en revue un certain nombre de confusions et d'énormités repérées dans la littérature des « ontologies ». Empruntons-lui ce rappel aux définitions :

« L'ontologie se définit […] comme la "science de l'Être" : elle reste constitutivement métaphysique, car la métaphysique est la "science de l'Être en tant qu'Être".

En revanche, le rôle des sciences reste précisément de rompre avec la métaphysique en définissant et en structurant de façon critique et réflexive des domaines d'objectivité. »

En nommant « ontologies » des nomenclatures terminologiques ou des thésaurus, et en affirmant que les termes qui y figurent, arrangés selon des principes qui feraient recaler à ses examens tout étudiant en linguistique, sont ainsi érigés au rang de concepts, ces cognitivistes balayent d'un revers de la main vingt-cinq siècles de pensée philosophique et deux siècles de linguistique pour ne nous donner en échange qu'un positivisme caricatural.

L'encre des paragraphes précédents à peine sèche, il convient de préciser ceci : il existe quelques logiciels parfaitement raisonnables, et même recommandables, dont le seul défaut est de se prétendre ontologiques. Il s'agit tout bonnement de carnets de notes électroniques et hypertextuels, qui permettent de gérer et d'indexer des données peu structurées, ce qui est utile, et l'informatique est bien adaptée à un tel usage. Et bien sûr le slogan « ontologie »

est plus éclatant que « bloc-note électronique », il faut bien le reconnaître. Se classent aussi dans cette catégorie des logiciels capables de comparer des sites Web consacrés à des thèmes voisins et d'établir entre eux des interliens vers des notions communes, c'est utile et intéressant, mais appeler cela le « Web sémantique » est pour le moins abusif : ici comme là, le péché véniel d'enflure verbale finit par confiner à l'imposture intellectuelle, ce qui est plus gênant.

Isabelle Boydens a écrit un livre (dérivé de sa thèse) [21] où, entre autres analyses pénétrantes dont nos chapitres précédents se sont déjà fait l'écho, elle traque dans les bases de données et les systèmes d'information des erreurs du même ordre que celles que nous évoquions quelques lignes plus haut. Elle y démonte des théories qui laissent croire qu'entre les symboles d'un langage et les objets d'un univers réel qu'il peuvent désigner il pourrait y avoir une correspondance isomorphe, c'est-à-dire telle qu'à chaque symbole corresponde sans ambiguïté un objet et un seul de cet univers, et que l'univers puisse être totalement décrit par le langage. Ainsi munie d'une armure à l'épreuve des syllogismes creux et des impostures techno-scientifiques, elle entreprend une analyse du système d'information de la sécurité sociale belge sur laquelle tout candidat à la construction d'un SI administratif ou de gestion devrait se pencher.

Nous croyons en avoir assez dit pour que le lecteur, surtout s'il a pris soin de consulter les ouvrages que nous avons cités, soit convaincu de l'impossibilité d'un isomorphisme entre quelque univers réel que ce soit et un système d'information quel qu'il soit. Un SI n'est par nature qu'une représentation, une abstraction qui ne retient à propos de la réalité qu'elle représente que les informations qui importent à l'objectif poursuivi, exprimées avec la précision nécessaire, mais pas plus. Cette conclusion ne vise pas à abolir tout espoir de construire un système d'information pertinent et utile, mais elle veut souligner les difficultés de l'entreprise et fixer les bornes des ambitions qu'elle pourra se donner.

Chapitre 8 Projets réussis et projets ratés

Sommaire

Le paysage des projets informatiques offre un spectacle varié de succès, d'échecs et surtout de demi-échecs et de succès partiels. Nous en présenterons ici quelques-uns qui offrent quelques leçons intéressantes.

8.1 Un directeur financier motivé

Au début des années 1980, époque antérieure à la généralisation des micro-ordinateurs, j'ai dirigé l'informatique (scientifique et de gestion) d'un petit établissement de recherche (150 personnes au total à l'époque) dont la gestion financière et comptable était assurée au moyen d'une machine électromécanique en fin de contrat d'entretien. Il fallait donc informatiser la gestion. Un appel d'offres fut lancé, et le fournisseur d'un logiciel spécialisé sélectionné. Le fournisseur du matériel informatique s'en déduisait, puisque ce logiciel ne pouvait fonctionner qu'avec un modèle d'ordinateur bien déterminé. Le tout fut livré et installé sans encombre. C'est là que les choses sérieuses commencent, et dans beaucoup de cas, comme j'ai pu observer, c'est à cet instant précis que l'échec ou le succès (en général l'échec) se décide – quand j'écris échec, on pourra m'objecter que la plupart des entreprises disposent d'un système de gestion informatisé qui fonctionne; sans doute, mais il s'agit souvent néanmoins d'un échec au moins partiel, parce que les délais n'ont pas été tenus, que les coûts ont explosé et que le service rendu est bien inférieur aux attentes initiales.

Lorsque l'on achète un progiciel de gestion, que ce soit comme ici le produit d'une petite société provinciale ou un prestigieux logiciel intégré à plusieurs millions d'Euros la licence, la complexité de l'univers décrit est du même ordre; une petite entreprise comme une grande a des fournisseurs, des clients, des salariés, des investissements, un capital, etc. Les bases de la comptabilité des entreprises ont été jetées à Sumer il y a près de cinq mille ans et depuis il y a eu des améliorations techniques mais pas vraiment de bouleversement conceptuel.

Le progiciel livré et installé par le fournisseur, aussi perfectionné soit-il, est une coquille vide : seuls les gestionnaires de l'entreprise sont en mesure de la remplir avec les données spécifiques qui en décriront le fonctionnement souhaité, conforme aux procédures de l'entreprise. Les données relatives directement à l'entreprise proprement dite, comme le

fichier des fournisseurs ou celui du personnel, ne posent pas de problème particulier. Il y a en revanche des données qui décrivent le fonctionnement du système de gestion lui-même, qui constituent un modèle abstrait de l'entreprise, que l'on peut considérer comme un méta-langage du langage interne de l'entreprise : ce sont notamment le plan comptable, les circuits de visas et d'habilitations (les « *work-flows* » en langage moderne), les agrégations d'informations et les états à produire, les synthèses pour les dirigeants et les responsables de secteurs. Ces données-là posent des problèmes beaucoup plus difficiles, parce que la détermination de leur forme et de leur contenu revient à décider du mode de gestion de l'entreprise.

Le lecteur des chapitres précédents devine déjà que dans un organisme public la locution « décider comment on gère l'entreprise » pose forcément un problème à un monde conçu pour que personne n'y décide jamais rien. Mais les entreprises privées éprouvent aussi des difficultés ! Si le progiciel est bien conçu, l'adapter à la structure particulière d'une entreprise se réalise par un paramétrage, avec un minimum de développements spécifiques si certaines fonctions n'ont pas été prévues, mais on cherchera à les éviter à tout prix. Les erreurs les plus courantes dans ce travail sont parmi les suivantes, d'après mes observations directes :

1. le paramétrage est confié à de simples exécutants dépourvus de vision d'ensemble et qui reproduisent leur routine habituelle et ses mauvaises pratiques en les gravant dans le marbre des règles du nouveau logiciel, ce qui crée une situation aggravée par rapport à la précédente ;
2. le dirigeant autocratique « qui a tout dans la tête » ne souhaite pas expliciter ses règles de gestion implicites pour les instancier dans les règles écrites d'un logiciel, ou il n'en est pas capable intellectuellement, et ainsi le système informatique ne correspondra jamais à la réalité de l'entreprise ;
3. l'entreprise n'a rien prévu pour le paramétrage et s'en remet aux options par défaut du logiciel : pour une petite entreprise ce peut être un moindre mal, mais c'est impraticable dès que la structure est un peu vaste et complexe ;
4. cas souvent combiné avec le cas 1 : en renfort des exécutants dépourvus de vision, évoqués au cas 1, on a fait venir des consultants pour le paramétrage ; chacun ajoute ses prescriptions dans le plus grand désordre ; on est ici dans un cas classique de prise de décision collective sans véritable responsabilité, les opérations de paramétrage vont enfler, prendre de plus en plus de temps, aboutir à une pagaille assez générale et à une masse de développements spécifiques souvent superflus.

Bref, pour que cela se passe bien, il faut que le véritable responsable de la gestion, si possible unique, s'implique personnellement dans le processus de paramétrage. Il faut qu'il soit capable d'avoir une vision suffisamment abstraite de son activité pour pouvoir la traduire en règles explicites de fonctionnement du système d'information, et doté d'une abnégation suffisante pour confier son pouvoir à cet outil impersonnel.

L'établissement de recherche qui nous occupe dans cet exemple était à l'époque (1982) pourvu d'un directeur financier qui possédait ces qualités. Non seulement il avait une connaissance approfondie de la comptabilité publique, mais il était capable de la décrire avec un niveau élevé d'abstraction, ce qui est essentiel pour un projet d'informatisation parce que cela permet d'isoler les processus significatifs et de les faire comprendre à l'informaticien de service, qui sinon croulerait sous les détails et passerait à côté de l'essentiel. Il avait d'ailleurs soutenu une thèse sur l'histoire de la comptabilité publique, à laquelle je dois notamment la référence à l'ordonnance du Vivier-en-Brie. En outre, il ne rechignait pas à acquérir les connaissances techniques dont il comprenait qu'elles lui permettraient de mieux maîtriser son système. Autant dire que son cas était une anomalie dans le contexte français ! Il faut dire qu'il était jeune et que son rang hiérarchique était modeste, il ne sortait même pas de l'ENA.

Le progiciel acheté reposait sur un système d'exploitation particulièrement bien adapté à des tâches de gestion, le système Pick créé par Richard Pick. Ce système était construit autour d'une base de données qui contenait toutes les données, que ce soient celles du système ou celles des utilisateurs. C'est-à-dire que chaque donnée était dotée d'un nom, d'un propriétaire, d'une liste d'utilisateurs autorisés et d'une durée de vie, et ce qu'elle soit en cours d'utilisation ou stockée pour un usage ultérieur : la distinction entre mémoire vive et mémoire de stockage (les « fichiers »), si contraire à l'intuition et si difficile à comprendre pour les néophytes, était donc abolie. Ainsi, le point d'indice, qui sert de référence pour le calcul des rémunérations de la fonction publique, existait de façon unique et bien identifiée dans le système, et lorsqu'une nouvelle valeur était publiée au Journal Officiel il suffisait d'une mise à jour pour que toutes les applications en tiennent compte. La simple idée d'un système aussi génial était insupportable pour le corporatisme des informaticiens, qui ont fini par avoir sa peau : il faut dire que Pick apportait une telle simplification à la gestion que sa généralisation aurait obligé quantité de programmeurs occupés à des tâches banales et répétitives, désormais réalisables par des non-informaticiens, à trouver un travail plus exigeant.

Pour accéder aux données de la base, les manipuler et produire des états, Pick fournissait un langage de requêtes, analogue à SQL[1] mais plus simple et plus expressif, baptisé avec une certaine outrecuidance *Français*. Notre directeur financier ne tarda pas à en posséder une maîtrise parfaite. La façon dont il procédait était la suivante : il me téléphonait vers 6 heures pour que je passe dans son bureau, et nous travaillions devant sa console jusque vers 21 ou 22 heures ; il m'expliquait ce qu'il avait fait et les obstacles qu'il avait rencontrés, presque toujours dus à son ignorance de quelque concept de base de l'informatique ; je lui expliquais la nature de l'erreur commise, et celle du concept qu'il devait assimiler pour la corriger, par exemple la notion de variable, celle d'affectation d'une valeur à une variable, celle d'écriture d'un enregistrement dans la base. Il était motivé, doué, l'exercice à résoudre le concernait au plus haut point, donc il comprenait vite.

Il nous suffit d'une trentaine de séances vespérales de ce type réparties sur une période de six mois pour qu'il rédige des procédures adaptées à toutes les fonctions de son service. Lors des dernières séances nous nous attaquâmes à un langage de plus bas niveau, *turing-équivalent*, *Basic*, pour effectuer plus efficacement quelques opérations qui atteignaient les limites des pouvoirs de *Français*.

Notre homme avait acquis ainsi une maîtrise exceptionnelle du travail de son service, il en connaissait intimement les moindres ressorts, puisqu'il les avait créés lui-même. Parmi les problèmes qu'il avait résolus, il y en avait bien sûr beaucoup qui découlaient des interactions entre les procédures qu'il avait rédigées lui-même et les fonctions de base incorporées dans le progiciel livré par le fournisseur, il avait donc acquis également une excellente connaissance de ce dernier, par la lecture de la documentation ou par l'interrogation tenace des employés dudit fournisseur. Bref, il disposait désormais d'un système parfaitement adapté aux besoins de l'établissement, qu'il était capable de faire évoluer au doigt et à l'œil quelle que soit la modification réglementaire catastrophique publiée par surprise au Journal Officiel.

En fait, dans cette expérience réussie d'informatisation, le directeur financier et moi-même avions appliqué les principes de l'*eXtreme Programming* sans le savoir, et pour cause. Je suis d'autant plus convaincu que c'est la seule façon raisonnable de développer un logiciel pour l'usage d'un tiers.

Ce système fonctionna à la satisfaction générale pendant cinq ou six ans. Puis il fallut en changer pour des raisons extérieures mais courantes dans le monde informatique : le constructeur de l'ordinateur utilisé se retirait de cette activité et arrêtait son service de

1 SQL *(Structured Query Language)* est un langage d'accès aux bases de données relationnelles normalisé par l'ISO et très répandu. Ce n'est pas un langage de programmation général (*turing-équivalent* selon le jargon que nous avons introduit au chapitre 4), mais il est de ce fait plus simple.

maintenance, et la société qui avait fourni le progiciel avait cessé son activité. Tout n'était pourtant pas désespéré : il existait d'autres fournisseurs du système Pick, et d'autres progiciels assez proches de celui que nous utilisions. Il aurait donc été envisageable de réitérer le processus suivi six ans plus tôt, qui n'avait somme toute été ni trop coûteux ni trop long. Mais la principale catastrophe était irrémédiable : notre directeur financier avait eu une promotion, méritée et due d'ailleurs en partie à son succès dans l'informatisation de la gestion, il était désormais directeur-adjoint de l'établissement et estimait que ce n'était pas à lui de faire ce travail de paramétrage. La responsabilité de l'informatisation fut confiée à un collaborateur du directeur financier, qui était loin d'avoir les aptitudes intellectuelles de son ancien chef, et qui s'entoura immédiatement d'une armée de personnes plus ou moins concernées mais qui avaient toutes en commun avec lui une aptitude modérée au travail à réaliser. Je n'ai pas eu connaissance des détails de la suite de l'histoire, mais je sais par les amis que j'ai gardés dans la maison qu'elle fut longue, triste, douloureuse et coûteuse : en un mot, conforme à toutes les autres expériences d'informatisation de la gestion qu'il m'a été donné d'observer de près.

8.2 L'Internet

8.2.1 Un réseau ouvert

S'il est un succès incontestable dans l'histoire des développements informatiques, c'est bien l'Internet, et il est donc intéressant de se pencher sur la façon dont ont été créés les logiciels d'ampleur considérable qui en assurent le fonctionnement.

Pour les moins informés de ses utilisateurs, l'Internet se confond avec le WWW et éventuellement avec le courrier électronique, qui n'en sont que deux usages parmi d'autres. Le WWW est un ensemble de méthodes destinées à permettre la publication sur l'Internet de documents indépendants de la nature du serveur et reliés entre eux par des liens, les URL *(Universal Resource Locator)*. Les URL (ces formules un peu étranges de la forme `https://laurentbloch.net/` qui apparaissent dans la barre d'adresses de votre navigateur) constituent un moyen très simple de désigner de façon précise et dépourvue d'ambiguïté un document quelconque sur le réseau.

L'Internet est un réseau de réseaux, un gigantesque système d'interconnexion que l'on peut comparer au système téléphonique international, même si ses principes de fonctionnement sont différents. C'est un système public : il suffit de s'abonner pour une somme modique, voir gratuitement, auprès d'un fournisseur d'accès (FAI) local pour avoir accès, le plus souvent sans paiement de redevances supplémentaires, à des serveurs dans le monde entier, ce qui n'est pas du tout le cas avec le système téléphonique international. De ce fait, l'Internet permet de publier toutes sortes de textes, d'images et d'autres informations qui seront accessibles à un public mondial, pour un coût minime. Toutes sortes d'informations dont la consultation demandait il y a dix ans de nombreuses heures de recherche dans des bibliothèques souvent accessibles aux seuls citadins des pays riches sont désormais au bout des doigts de tout un chacun.

Si la grande presse a pu faire ses choux gras des abus de cette liberté, et si la recherche d'informations sérieuses aux dépens des moins sérieuses demande, comme par le passé, une bonne connaissance de son domaine de recherche, l'arbre ne doit pas cacher la forêt, et l'existence de ce moyen de diffusion universel et bon marché représente une véritable étape dans l'histoire de la pensée et de la civilisation. Son existence a permis l'apparition d'innombrables sources de créativité dans les domaines les plus variés, et qui n'auraient sans lui jamais trouvé la voie de la publication. On trouvera dans le livre de Lawrence Lessig [75] une analyse approfondie du rôle d'un système de publication libre d'accès, et des menaces qui pèsent aujourd'hui sur lui.

8.2.2 Modèle en couches et principe périphérique

La charpente de l'Internet (comme de toute architecture de réseau) est constituée de protocoles, c'est-à-dire de documents qui fixent les règles imposées à un type de communication donné. Les protocoles de réseau sont généralement représentés selon une architecture en couches imaginée par le chercheur néerlandais Edsger Wybe Dijkstra (1930–2002) dans un article fameux publié en mai 1968 par les CACM *(Communications of the Association for Computer Machinery)*, « *The structure of the THE multiprogramming system* » [38].

Avant d'être le plan de construction du réseau concret, l'architecture en couches est un modèle destiné à se représenter intellectuellement les choses par des abstractions. Ce modèle est utile pour « penser un objet dans lequel plusieurs logiques s'articulent » (Michel Volle, `http://www.volle.com/opinion/couches.htm`), lorsqu'il faut séparer différents niveaux d'abstraction. On nommera *couches basses* celles qui décrivent la transmission de signaux sur des supports physiques tels que câble téléphonique, faisceau hertzien ou fibre optique, tandis que les couches intermédiaires concernent l'acheminement de messages complexes à travers des réseaux à la topologie également complexe, et que les couches hautes traitent de la présentation de ces messages à travers une interface utilisateur, de l'identification de leur destinataire et de son authentification. Les ensembles de règles et de conventions qui régissent les communications entre les couches de même niveau de plusieurs systèmes communicants constituent des *protocoles de communication*. Les règles et les conventions qui régissent les échanges entre une couche donnée et la couche immédiatement inférieure d'un même système constituent une *interface*.

Une des choses qui a fait la force de l'Internet, c'est le principe *« end to end »*, que l'on pourrait traduire, avec Solveig Godeluck [51], par le *principe périphérique*, ou par « l'intelligence est aux extrémités », c'est-à-dire dans les programmes installés sur les ordinateurs des utilisateurs finals ; l'architecture du réseau reste la plus simple possible, on s'interdit d'introduire au sein des infrastructures des dispositifs ou des optimisations spécifiques pour tel ou tel usage particulier. Pour implanter un nouveau protocole qui devra permettre tel nouveau mode de communication entre deux ordinateurs connectés au réseau, par exemple la téléphonie par l'Internet, tout ce qu'il faudra, c'est que ces ordinateurs soient dotés chacun des protocoles de base (*Internet Protocol*, ou IP, et en général *Transmission Control Protocol*, ou TCP), et bien sûr y installer les logiciels propres à la nouvelle application. Toutes les fonctions nouvelles seront donc implantées dans les ordinateurs aux extrémités de la communication, sans nécessiter aucune adaptation du réseau et des nœuds intermédiaires. C'est-à-dire, par exemple, que lorsque Tim Berners-Lee a voulu créer HTTP, le protocole du WWW, il n'a eu à demander l'autorisation de personne, parce que ce nouveau protocole ne nécessitait aucune modification de l'infrastructure du réseau – bien sûr, pour que HTTP devienne un standard de l'Internet, il aura fallu ensuite quelques formalités, que nous évoquerons ci-dessous. Ce principe *« end to end »* garde l'infrastructure du réseau aussi simple, ouverte et favorable à l'évolution que possible.

Le fait que le contrôle de l'accès à l'Internet ne puisse être accaparé par personne garantit à ses usagers qu'ils pourront l'utiliser sans restriction tant pour y publier que pour y consulter des informations, et c'est ce qui a fait son succès, face à l'incompréhension totale des grandes entreprises de télécommunications comme de télédiffusion, qui continuent à n'y voir qu'un hybride entre TF1 et le catalogue de la Redoute. Tant pis pour elles.

8.2.3 Mise au point historique

Entamons cet examen par la réfutation d'une légende aussi fallacieuse que répandue et répétée *ad nauseam* : **l'Internet n'a pas été créé selon un cahier des charges rédigé par les militaires américains en vue de préserver une capacité de communication après une frappe nucléaire**. Il n'y a jamais rien eu qui ressemble de près ou de loin à un « cahier

des charges de l'Internet ». Cette thèse télescope plusieurs événements liés mais distincts. Paul Baran, du *think tank* RAND, contractant du DoD *(Department of Defense),* avait posé les principes d'un système de communications dépourvu de point de centralisation unique afin de maintenir certaines liaisons même si certains nœuds venaient à être détruits. Les travaux de Baran furent publiés entre 1960 et 1964. Le même DoD, plusieurs années plus tard, en 1969, a impulsé par son agence ARPA *(Advanced Research Projects Agency)* la création du réseau ARPANET, qui n'était pas destiné aux communications militaires mais à faciliter la collaboration entre centres de recherches universitaires et industriels sous contrat avec l'ARPA. À sa création ARPANET reliait quatre universités : Stanford à Palo Alto, les campus de Los Angeles et de Santa Barbara de l'Université de Californie, et Utah à Salt Lake City. On pourra consulter à ce propos le livre passionnant de Katie Hafner et Matthew Lyon [56].

C'est en 1974 que Vinton Cerf (de l'université Stanford) et Robert Kahn de BBN (Bolt, Beranek & Newman) publièrent le premier article sur TCP/IP. En 1976, la clairvoyance de Vint Cerf et de Robert Kahn, le financement de BBN et les contrats de l'ARPA devenue entre temps la DARPA donnèrent naissance au protocole réseau de l'Internet, destiné à devenir TCP/IP. En 1977 ce protocole commença à fonctionner, et c'est de ce moment que l'on peut dater la naissance de l'Internet expérimental.

Nous avons déjà mentionné comme une innovation propre à l'Internet le principe périphérique *(« end to end »)*, nous pouvons en ajouter une autre, liée à la précédente et qui a trait à la fiabilité. Voilà de quoi il s'agit : on dit qu'un protocole de transmission est *fiable* s'il comporte une vérification que les données émises ont été transmises sans erreur ; la vérification a lieu au moyen d'informations redondantes ajoutées aux données émises, dont on vérifie à l'arrivée la cohérence avec le corps du message. Il est en effet dans la nature des communications à distance d'être sujettes aux erreurs de transmission. Si un message a été corrompu lors de sa transmission, il faut en général prévenir son émetteur pour qu'il le réémette. Si l'on dispose d'un protocole fiable pour faire circuler des données entre deux nœuds adjacents, et si l'extrémité réceptrice de la communication vérifie l'intégrité des données, conformément au principe de l'intelligence aux extrémités, il est inutile que le protocole intermédiaire chargé de faire circuler les données dans le réseau (la *couche réseau*) comporte lui-même un contrôle d'erreur. Ce principe d'une *couche réseau non-fiable*, formulé pour la première fois par Louis Pouzin pour le réseau français Cyclades, permet une grande simplicité du protocole de réseau.

C'est en 1978 pendant une pause d'une conférence à Marina del Rey que Vint Cerf et Jon Postel tirèrent les conséquences de ce principe de la couche réseau non-fiable en divisant leur protocole en deux parties : un protocole fiable de transport de bout en bout, TCP, et un protocole réseau non-fiable, IP. Cette innovation hardie allait constituer une des forces de l'Internet, avec quelques autres qu'il n'est pas possible d'exposer ici et pour lesquelles on se reportera avec profit au livre de Katie Hafner et Matthew Lyon [56].

Un peu plus tard, en 1979, le groupe de recherche en système *(Computer Systems Research Group, CSRG)* de l'Université de Californie à Berkeley allait incorporer TCP/IP à une nouvelle version d'Unix, dite BSD *(Berkeley Software Distribution)*.

En 1982 la Défense américaine adopte officiellement les protocoles TCP et IP (Internet Protocol) pour le réseau ARPANET et accepte de les distribuer gratuitement sur le réseau. C'est ainsi que s'est nouée une situation qui allait unir les destins de l'Internet, d'Unix et des logiciels libres.

Tous ces éléments ont contribué à donner naissance en 1986 à l'Internet tel que nous le connaissons, quand les centres de recherche connectés à ARPANET ont voulu communiquer avec les universités de leur choix et que la NSF *(National Science Foundation)* a entrepris de leur en donner les moyens en créant son propre réseau, *NSFNet*.

Dès le début des années 1980 ARPANET acceptait des connexions à partir d'universités non américaines. Il s'agissait souvent (dans le cas de la France notamment) de liaisons

indirectes, établies par des passerelles et des relais complexes, mais qui donnaient accès au réseau ; la France eut accès au courrier électronique ainsi en 1984, mais ce n'est que le 28 juillet 1988 que notre pays fut vraiment raccordé à l'Internet par une liaison permanente en TCP/IP [57].

8.2.4 Organisation administrative de l'Internet

Il faut rappeler ici que l'Internet n'est la propriété de personne ni d'aucune institution, ce qui semble parfois difficile à comprendre.

Nous invitons le lecteur à quelques secondes de réflexion admirative devant un dispositif technique conçu à l'origine pour un réseau d'une centaine de nœuds et qui a pu résister à une croissance de six ou sept ordres de grandeur, d'autant plus que cette conception n'était le fait ni d'un grand groupe industriel ou financier, ni d'un gouvernement, ni d'un conglomérat de telles puissances. Mais peut-être était-ce là le secret ?

Le fonctionnement de l'Internet, à l'image de sa construction, repose sur la coopération volontaire. Nous donnons ici l'organigramme général de son organisation :

- L'*Internet Architecture Board (IAB)* est responsable des grandes orientations et de la coordination.
- L'*Internet Engineering Task Force (IETF)* se charge de la normalisation à court terme et émet les *Requests for Comments (RFC)*, qui sont les documents de référence pour le fonctionnement du réseau [2]. Toutes les RFC sont accessibles par l'URL *(Universal Resource Locator)* `http://www.ietf.org/rfc/` ou sur de nombreux sites miroirs. Nous ne saurions trop en conseiller la lecture : même si leur qualité littéraire est inégale elles fournissent sur l'Internet une information de première main, souvent exposée très clairement.
- L'*Internet Steering Group (IESG)* coordonne l'IETF, dont l'effectif est devenu très important.
- L'*Internet Assigned Numbers Authority (IANA)* centralise et contrôle les conventions relatives à l'identification des objets du réseau, et notamment veille à l'unicité des adresses.
- L'*Internet Corporation for Assigned Names and Numbers (ICANN)* , créée en 1998 à l'initiative du gouvernement américain, supervise l'attribution des noms de domaines et des adresses.

Ainsi les décisions organisationnelles et techniques sont prises par des instances aux séances desquelles tout un chacun peut assister et participer, les séances des instances de pilotage (*IAB, IESG, ICANN*) étant toutefois réservées à leurs membres élus. Cette organisation coopérative ne signifie pas l'absence de rapports de force marchands ou politiques, mais elle exclut (au moins à court terme) la prise de contrôle par une entreprise unique. L'*ICANN* soulève bien des critiques du fait de l'influence déterminante qu'y exerce le gouvernement américain, mais il est significatif que la réaction à ce qui pourrait devenir une mainmise est rendue possible par la structure ouverte des autres instances, par contraste avec ce qui se passe à l'ISO ou à l'UIT (Union Internationale des Télécommunications).

8.2.5 Organisation topographique de l'Internet

La figure 8.1Topographie simplifiée de l'Internetfigure.8.1 donne une représentation schématique de la topographie de l'Internet. Cette représentation est simplifiée, notamment elle est purement arborescente, alors que rien n'empêche une entreprise d'avoir plusieurs fournisseurs d'accès à l'Internet (FAI), ni un FAI d'être connecté à plusieurs centres d'in-

2 Citons ici le nom de Jon Postel, éditeur des RFC depuis la première en 1969 jusqu'à sa mort en 1998, et auteur ou coauteur de 204 d'entre elles, ce qui lui a conféré une influence considérable sur la physionomie du réseau.

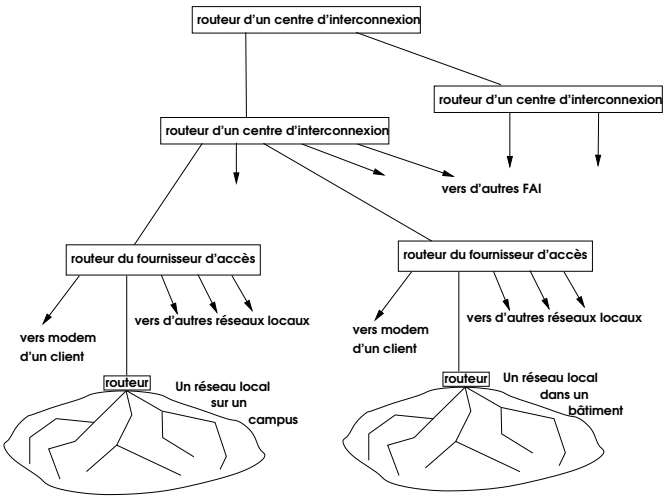

Figure 8.1 : Topographie simplifiée de l'Internet

terconnexion de niveau supérieur, ce qui complique le schéma et le transforme d'un arbre en un graphe connexe quelconque. Chaque nœud (ordinateur connecté, ou autre type d'équipement) est identifié par une adresse (dite *adresse IP*) unique à l'échelle mondiale, un peu comme un numéro de téléphone précédé de tous ses préfixes nationaux et régionaux. L'essentiel dans l'établissement d'une communication, c'est, à chaque nœud, de savoir quel est le prochain nœud sur l'itinéraire, et par quelle liaison l'atteindre. Les nœuds terminaux, origines ou destinations des communications, sont au sein des réseaux locaux de campus ou d'entreprise. Un routeur est un nœud particulier, doté de plusieurs interfaces réseau (et donc de plusieurs adresses), ce qui lui permet d'être connecté simultanément à plusieurs réseaux et de faire passer les paquets de données d'un réseau à un autre, en fonction de *règles de routage* qui auront été enregistrées dans sa mémoire.

8.2.6 Architecture de l'Internet

Les documents qui décrivent les protocoles – et par conséquent l'architecture – de l'Internet s'appellent des *Requests for Comments (RFC)* [58]. Les protocoles les plus connus sont IP *(Internet Protocol)* pour l'acheminement d'un paquet de données à travers un réseau complexe, TCP *(Transmission Control Protocol)* pour le transport d'un segment de données de bout en bout selon une connexion établie, puis au niveau applicatif SMTP *(Simple Mail Transfer Protocol)* pour le courrier électronique, HTTP *(Hypertext Transfer Protocol)* pour le WWW, FTP *(File Transfer Protocol)* pour le transfert de fichiers, etc.

Les protocoles sont publiés par l'IETF à partir de soumissions spontanées. Le RFC 2026 définit la procédure de validation des RFCs : une soumission doit suivre un processus nommé « *standards track* », qui comporte trois étapes : « *Proposed Standard* », « *Draft Standard* », et « *Standard* ». L'introduction d'un document dans ce processus est effectuée à l'initiative de l'IESG, qui choisit les « *Proposed Standards* » parmi des documents nommés « *Internet-Drafts* », soumis sans formalités à la discussion. Pour qu'une soumission soit examinée avec quelque chance de succès il est de coutume qu'elle soit accompagnée d'un prototype de logiciel pour l'illustrer.

Il est clair que la spontanéité des soumissions est variable, et, au fur et à mesure que l'Internet devenait un enjeu économique important, les industriels de l'informatique et des réseaux ont contribué de plus en plus intensément à la création de RFCs, mais sans que le caractère consultatif et concerté du processus soit fondamentalement altéré.

De fait, aujourd'hui, le fonctionnement de l'Internet repose sur des documents de normalisation librement accessibles à tous, contrairement aux normes ISO qui ne sont accessibles que moyennant finances. Faire payer ou non l'accès aux normes, c'est une décision lourde de conséquences, qui définit le public auquel on s'adresse. Des logiciels libres sont disponibles pour tous les usages de l'Internet : serveurs de messagerie *(Sendmail, Postfix, Qmail, Exim)*, serveurs WWW *(Apache)*, serveurs de noms *(Bind, Djbdns, Unbound)*, navigateurs *(Mozilla)*... Ce processus de création par accrétion d'une infrastructure mondiale, à la cohérence extrême, utilisée par des centaines de millions de personnes, en l'absence de tout pilotage centralisé, constitue un défi réellement fascinant pour tous les architectes de systèmes.

8.2.7 L'Internet est-il un système d'information ?

Le lecteur pourrait s'interroger sur le bien-fondé de la mention de l'Internet dans un ouvrage consacré aux systèmes d'information. La réponse est oui, l'Internet peut être vu comme un S.I. de plusieurs façons : d'abord, l'Internet incorpore dans son infrastructure plusieurs systèmes d'information nécessaires à son fonctionnement. Citons le système de noms de domaines (*Domain Name System*, ou DNS), qui permet, si l'on connaît le nom d'un serveur, de trouver l'adresse qui permettra de l'atteindre, et vice-versa. Fonctionnellement, le DNS est extrêmement simple : c'est une table à deux colonnes, dans la colonne de gauche des noms, dans la colonne de droite les adresses correspondantes. Techniquement, c'est une immense base de données distribuée sur l'ensemble de la planète, avec de fortes contraintes de disponibilité et de mise à jour, peut-être la base la plus grande et la plus complexe qui existe. L'ensemble des informations de routage, qui permettent l'acheminement des communications d'un bout à l'autre de la planète, complète ce système d'information.

L'Internet est aussi un S.I. d'un autre point de vue : lorsque l'on écrira l'histoire de la pensée à la fin du XX^{e} siècle, nul doute qu'y figure l'invention de l'URL *(Universal Resource Locator)* et des moteurs de recherche, qui ont révolutionné les méthodes de recherche d'information, en abolissant toutes sortes de barrières matérielles et en réduisant de façon drastique les délais d'accès à un volume de documents inimaginable il y a vingt ans.

8.3 Socrate à la SNCF

La SNCF a mis en service au début de 1993 un nouveau système de réservation baptisé Socrate. Lors du lancement de ce projet l'objectif poursuivi était, pour citer le rapport de la Cour des Comptes [32], « le lancement d'un nouveau système de distribution commerciale : à la fin des années 1980, dans la perspective du développement du réseau de lignes à grande vitesse et de l'accentuation de la concurrence avec le transport aérien, il était devenu vital pour la SNCF d'entrer comme partenaire dans un système global de distribution, afin de garantir l'égalité de traitement du chemin de fer par rapport à l'avion dans les agences de voyages, qui représentent plus de 20% de sa distribution commerciale. »

La SNCF escomptait une augmentation de 50% de son trafic voyageurs entre 1986 et 1995, et dans une telle perspective le système de réservation RESA dont elle disposait à l'époque était jugé désuet ; il fallait soit le transformer soit le remplacer. En 1988 la SNCF choisit d'abandonner le système RESA et « de se rapprocher de la société AMR, filiale d'*American Airlines*, dont le système de distribution SABRE occupait la première place dans le transport aérien mondial. »

En fait, derrière cette décision de la SNCF peuvent se lire rétrospectivement plusieurs orientations qui ne se situent pas toutes sur le même plan et qui ne sont pas forcément liées entre elles :

— Conformément au but initial, offrir aux agences de voyage un système de réservation comparable à celui des compagnies aériennes.

— Modifier la politique tarifaire selon des principes nouveaux qui seront décrits ci-dessous, pour accroître les recettes commerciales.
— Infléchir ses relations avec sa clientèle vers un alignement sur les principes du marketing, en s'écartant progressivement des principes du service public, notamment par le recours à la segmentation de la clientèle. La segmentation consiste à essayer d'attirer et de « fidéliser » la partie de la clientèle jugée la plus rentable, et à imposer des contraintes supplémentaires au reste de la clientèle, jugée moins intéressante.
— Diminuer la dépendance de l'entreprise à l'égard de son service informatique interne en faisant appel à un prestataire extérieur.

Cette volonté multiforme de modernisation doit être appréciée dans le contexte des années 1980, marquées par de graves accidents et de grandes grèves qui ont mis en lumière la vétusté de pans entiers tant de l'infrastructure technique que de la réglementation et de la situation sociale.

SABRE était un système éprouvé mais assez ancien, puisqu'il avait été conçu en 1960 par *American Airlines* et IBM. Il comptait de nombreuses références dans le monde du transport aérien, mais aucune dans celui du rail : pour espérer un succès à la SNCF, il aurait été nécessaire de l'adapter aux caractères propres du transport ferroviaire, et l'ampleur de ces modification avait été fortement sous-estimée lors de la phase de définition et de lancement du projet, ce qui amènera à les réaliser en catastrophe après l'ouverture du système au public.

Le lancement du système en 1993 s'est très mal passé : on peut dire sans abus que des trains circulaient en partie vides cependant que les voyageurs qui auraient souhaité les prendre restaient à quai faute d'avoir pu obtenir un billet. Les causes de cet échec étaient multiples :

— le personnel ne savait pas utiliser le système ;
— les bases de données qui décrivaient les trains, les itinéraires, les correspondances, les horaires et les tarifs étaient défectueuses ;
— la clientèle était déroutée par la nouvelle politique commerciale et tarifaire, qui donnait une impression négative d'augmentation des coûts ;
— les équipes informatiques de la SNCF ne connaissaient pas suffisamment Socrate pour en corriger les défauts.

Les dysfonctionnements du système Socrate ont été importants, ils ont entraîné pendant plusieurs années consécutives une diminution sensible de la fréquentation des trains par la clientèle. Il est certain qu'une entreprise privée non subventionnée aurait eu le plus grand mal à se relever d'un tel échec, qui n'a pu être compensé que par une ponction importante dans les finances publiques. Le rapport de la Cour des Comptes [32] est éloquent à ce sujet. On pourra également consulter les minutes d'un débat [69] organisé sur la Cinquième chaîne de télévision, et bien d'autres documents accessibles sur le WWW.

Le cas de Socrate est intéressant parce qu'il est devenu public, il a pu être analysé par des organismes de contrôle gouvernementaux et par des experts indépendants, et ces analyses ont été rendues publiques. Lorsque de tels dysfonctionnements surviennent dans une grande banque, et il en survient, ils restent le plus souvent confidentiels.

Il ressort assez clairement de la lecture des documents publiés que, dans l'échec de Socrate, les causes techniques étaient secondes par rapport aux causes afférentes à une mauvaise gestion du projet. Il suffit de regarder les rubriques du rapport de la Cour des Comptes :

— La SNCF a inversé en cours de réalisation les priorités du projet, initialement destiné à permettre aux agences de voyage de réserver des places de train aussi facilement que des places d'avion, l'orientant soudain vers un objectif de maximisation des recettes commerciales.
— Les problèmes posés par l'alimentation du nouveau système à partir des « données voyageurs » ont été gravement sous-estimés ; par exemple, le nombre de relations

par correspondance introduites dans Socrate n'était que de 4 000, alors qu'il en aurait fallu au moins dix fois plus.

— De nouveaux principes de tarification, qui ont nécessité une modification en catastrophe du cahier des charges de la SNCF, ont été introduits furtivement : le système précédent, dit « de péréquation », permettait à un usager de se présenter à un guichet SNCF pour faire l'achat d'un titre de transport qui lui garantissait, en fonction du prix payé, d'être transporté sur le même nombre de kilomètres quel que soit le moment et quelle que soit la ligne sur laquelle il devait se déplacer. La SNCF a décidé de faire disparaître la péréquation au profit d'un ajustement du rapport offre / demande ; il faut désormais gérer une offre complètement différente avec des gammes de prix variables, train par train et jour par jour ; il faut également appliquer des critères de tarification variables en fonctions des caractéristiques des voyageurs : âge, fréquence de voyage, etc. Il s'agit aussi de pouvoir traiter les services associés aux billets, comme la réservation d'hôtel à l'arrivée et la location d'une voiture par exemple.
— La sensibilité de la clientèle aux prix du transport a été mal appréciée.
— La formation du personnel de vente à l'exploitation du système Socrate a beaucoup laissé à désirer, notamment pour ce qui est de l'apprentissage opérationnel.
— Le dialogue de la SNCF avec les associations d'usagers est resté trop strictement formel, et la communication vers le grand public a été défaillante.
— L'avenir de l'exploitation du système a été longtemps hypothéqué par une situation de dépendance de l'entreprise vis-à-vis de son fournisseur et par des difficultés à maîtriser la politique tarifaire.
— Le coût total de réalisation aura été en fait de l'ordre de 2,1 milliards de francs 1988, soit plus du double du devis présenté lors du lancement du projet.

Les commentaires de certains experts indépendants sont également instructifs ; ainsi Daniel Faita, chercheur au Centre de recherches « Analyse pluridisciplinaire des situations de travail » de l'Université de Provence, co-responsable de l'expertise « nouvelles technologies » pour le projet Socrate, a déclaré à Nathalie Le Breton [69] :

« Auparavant, les stratégies de vente des agents SNCF consistaient à repérer puis isoler dans la demande de l'usager les éléments permettant de réaliser une transaction à la fois satisfaisante pour l'usager et efficace en terme de prestation. En fonction de la logique de Socrate, le vendeur ne devait plus laisser libre cours à l'expression de l'usager mais guider le dialogue en lui proposant des disponibilités. C'est alors l'offre qui prévaut sur la demande...

[La démarche suivie] a abouti à ignorer que, dans les activités commerciales de la SNCF, existaient des données fondamentales – comme la gestion du temps qui est un critère essentiel dans toute activité d'interaction – qui ont été complètement simplifiées par la nouvelle politique commerciale de la SNCF. L'approche techniciste du projet conduisait à faire du gain de temps dans les transactions l'objectif central des réformes. Or, ceci était en contradiction avec l'une des dimensions fondamentales de cette activité commerciale...

Avec Socrate, c'est le temps employé par le système qui sert à prescrire l'évaluation du temps de transaction et non pas le temps humain qui lui intègre réflexion et interrogations et qui permet de mieux aboutir au terme de la transaction. »

Ce dernier paragraphe évoque exactement le passage de Philippe Zarifian que nous citions à la page 26 pour caractériser l'organisation taylorienne du travail, laquelle se révèle aussi peu adaptée à la démarche commerciale qu'à la réalisation de systèmes informatiques.

Sur le plan technique, Socrate est une adaptation du logiciel SABRE développé pour *American Airlines*. L'ampleur des conséquences à tirer des différences entre transport aérien et transport ferroviaire a été nettement sous-estimée. Les équipes informatiques de la SNCF n'étaient pas en mesure d'intervenir sur le logiciel, puisque tout était externalisé. Daniel Faita note :

« Je ne voudrais pas que vous retiriez de mon exposé une impression exclusivement caricaturale. Ceci dit, il faut bien admettre que les informaticiens de la SNCF n'ont pas été impliqués, ni dans le choix du logiciel, ni dans son adaptation. Le choix a été largement guidé par des considérations théoriques qui ont ignoré de manière assez nette, d'une part, la logique de l'adaptation de l'outil et, d'autre part, les impératifs de soumission aux normes techniques propres aux transports ferroviaires. »

Bref, le projet Socrate semble bien avoir été marqué par le fantasme de l'achat de progiciel tout prêt, en négligeant la nécessaire participation des membres de l'entreprise à son adaptation aux situations concrètes.

Le soin de déterminer si les leçons de Socrate ont profité au site WWW de la SNCF, le plus fréquenté de France, est laissé au lecteur.

Conclusion

Nous avons essayé de montrer que, avant de réfléchir au pourquoi et au comment d'un projet informatique, il faut avoir à l'esprit un certain nombre de caractères propres au logiciel : il est abstrait, invisible et infiniment malléable ; lors de son élaboration, pour prendre le contre-pied d'une métaphore usuelle mais fallacieuse qui compare la réalisation de logiciels à celle de bâtiments, il sera possible de modifier le plan et les fondations au moment d'attaquer le deuxième étage ! En outre, le logiciel est complexe : cette complexité est dans sa nature ; un logiciel simple, au sens où par exemple il serait constitué par la répétition d'un petit nombre d'éléments identiques, n'aurait aucun intérêt.

L'idée que la programmation n'est pas une fabrication, mais un acte de conception, et des plus complexes, découle de la nature abstraite du logiciel ; cette idée est au centre de ce livre, comme le lecteur aura pu le constater. On peut le dire autrement, comme Paul Graham [53] par exemple : « En programmation, comme dans beaucoup d'autres domaines, la difficulté n'est pas tant de résoudre les problèmes que de choisir les problèmes à résoudre ». Il s'agit là de la programmation entendue dans un sens assez large, qui englobe aussi bien le paramétrage d'un progiciel que l'assemblage de plusieurs logiciels pour réaliser une application ; on aurait tort de prendre à la légère de tels travaux, ils posent des problèmes intellectuels du même ordre que l'écriture d'un logiciel à partir d'une feuille de papier blanc.

De cette réhabilitation du travail de programmation, souvent dévalué ou sous-estimé, découle une conséquence qui ne paraîtra décourageante qu'aux porteurs d'espoirs déraisonnables : la programmation n'est pas industrialisable, elle est vouée à rester un artisanat. Les méthodes de cet artisanat pourront progresser, grâce par exemple à de meilleurs langages, à de meilleurs environnements de développements, à des systèmes de preuve plus puissants qui déplaceront l'effort du programmeur plus loin du concret technique et plus près du problème abstrait, mais la nature fondamentale de son activité demeurera artisanale. Cela n'est pas dû à de mauvaises méthodes de travail et d'organisation, dont l'amélioration ouvrirait la voie à l'industrialisation : ce caractère artisanal est intrinsèque à la programmation, et je soutiens l'idée que cet état de fait tient à des raisons fondamentales, inhérentes à la nature de la pensée humaine et à celle de l'activité de programmation des ordinateurs.

L'adoption de ce point de vue conduit logiquement à abandonner un certain nombre de pratiques, dont je me suis employé à analyser les faiblesses : organisation d'une division taylorienne du travail informatique, externalisation massive (comme s'il s'agissait du nettoyage et du gardiennage des locaux), constitution de grandes équipes organisées en ateliers dans l'espoir de réaliser des économies d'échelle comparables à celles que la grande industrie des XIX^{e} et XX^{e} siècles a obtenues dans des contextes et des domaines très différents. Il faut aussi renoncer aux grandes visions bureaucratiques de gestion de l'activité de développement : méthode canonique de gestion de projets, démarche qualité conforme à ISO 9000.

Examiner ces questions nous a mené à un regard critique sur deux tendances de fond des sociétés développées depuis un siècle et demi au moins : l'accroissement tentaculaire de la bureaucratie et les abus de la division du travail. Ces deux mouvements ont depuis longtemps déjà entraîné la sclérose des services publics de l'État, du moins en France ; les grands groupes industriels et financiers ne sont pas non plus à l'abri de ce phénomène. Cette bureaucratie des grandes structures fait peser un poids énorme sur l'ensemble de la société, qui se répercute particulièrement sur les petites et moyennes entreprises. Les structures

de la recherche scientifique et de l'enseignement ne sont pas épargnées par ces maladies modernes, dont un des symptômes est l'*imitation de travail* décrite de façon approfondie par Alexandre Zinoviev [134] et que nous avons évoquée à la page 22.

Pourquoi ces questions relèvent-elles de la problématique du système d'information ? Par deux aspects :

— le système d'information des entreprises est au cœur des appareils productifs contemporains, et par là il fournit une contribution majeure aux formes de la bureaucratie et de la division du travail ;
— les maîtres d'ouvrage des systèmes d'information ont tenté, de diverses façons, d'*appliquer* ces mêmes formes de la bureaucratie et de la division du travail à la construction de systèmes d'information.

C'est surtout le second de ces deux aspects, l'effort pour appliquer des principes tayloriens et bureaucratiques au développement informatique, qui a fait l'objet du présent ouvrage ; nous avons essayé d'illustrer la vanité de ces tentatives, et leur échec.

Si le développement informatique ne se prête pas à l'industrialisation et doit rester artisanal, il convient d'en tirer les conséquences sur le plan de l'organisation. Les grands projets menés avec des effectifs importants organisés selon une hiérarchie complexe accumulent les occasions d'échec, ce que l'expérience confirme. Les projets de taille moyenne menés par de petites équipes très intégrées ont de bien meilleures chances de succès. Il est donc judicieux de se rapprocher le plus possible de ce modèle d'organisation. Pour ce qui est des rapports avec le client final ou son représentant, il n'y a que deux hypothèses viables : soit il s'implique totalement dans l'équipe, y compris jusqu'à un certain point dans les aspects techniques (scénarios, tests, validation), soit il reste à distance. Le client qui chaque lundi matin téléphone pour communiquer à l'équipe de développement les modifications de spécifications dont il a eu l'idée pendant le week-end conduit son projet à un échec certain, ce qui n'empêche pas ce comportement d'être très courant. Or justement les spécifications changent, parce que les circonstances de la vie changent ; deux conséquences en découlent :

— L'implication forte du maître d'ouvrage ou de son mandataire est indispensable si l'on veut développer un système adapté à un besoin du client (par opposition à un logiciel à usage général). Les logiciels dont il est question ici sont pour l'essentiel ceux que nous avons rangés dans la classe 1 définie par notre introduction (page 6), c'est-à-dire ceux qui sont destinés à assurer le fonctionnement des systèmes d'information des entreprises.
— Il faut privilégier les méthodes de développement incrémentales, dont nous avons cité un exemple avec *eXtreme Programming*, mais il en existe d'autres.

Les grands projets de gestion offrent le paysage désolé de taux d'échec élevés. Il faut considérer qu'un système comme Socrate à la SNCF, qui finit par marcher après qu'aient été multipliés par deux le budget prévisionnel et le délai de réalisation, non sans avoir entraîné plusieurs années de désordre dans l'entreprise, est un échec. De nombreux échecs de ce type ne sont pas ébruités si la dissimulation est possible. Le cas de Socrate a été l'objet de débats publics et d'analyses détaillées par des experts indépendants, ce qui permet de savoir que les facteurs techniques n'ont compté que pour une part, et pas la plus importante, dans les errements du projet. L'absence de stratégie claire, l'intervention désordonnée de plusieurs groupes de pouvoir au sein de l'entreprise, l'insuffisance de la formation dispensée aux personnels concernés ont compté sans doute plus que les défauts de l'adaptation du logiciel au besoin.

Les projets de notre classe 1 (systèmes d'information de gestion) rencontrent plus souvent l'échec que ceux de la classe 2 (systèmes techniques) ou de la classe 3 (infrastructures informatiques). Nous avons évoqué plusieurs facteurs de nature à expliquer cet état de fait : les systèmes de la classe 1 sont techniquement plutôt moins complexes que ceux des classes 2 et 3, mais ils sont beaucoup plus complexes socialement.

Pour les projets des classe 2 et 3, tous les participants appartiennent au même univers intellectuel, ce sont pour l'essentiel des ingénieurs qui parlent le même langage, et qui se reconnaissent une obligation morale de comprendre les problèmes techniques, même ceux qui ne sont pas exactement de leur spécialité.

Si l'on considère un projet typique de classe 2, comme un nouveau modèle d'avion ou une ligne de métro sans conducteur, le coût du logiciel représente une faible partie du budget total, cependant que son dysfonctionnement peut provoquer le naufrage complet du projet ; de ce fait, le maître d'ouvrage ne reculera pas devant des méthodes de développement coûteuses et le recours à des équipes de haut niveau. Nous serions tentés de dire que les entreprises industrielles qui n'ont pas compris cela ont disparu ou sont condamnées à brève échéance, parce qu'un avionneur, par exemple, ne peut se dissimuler que le logiciel de pilotage de son prochain avion doit être au cœur de ses préoccupations. En outre, une fois qu'un projet industriel est lancé, il est hors de question d'en changer les spécifications tous les deux mois : en effet tout le monde comprend bien qu'on ne peut pas passer son temps à démolir et à reconstruire les usines.

Nous avons brièvement décrit à la page 124 un système de classe 3 de première grandeur : l'Internet. Le projet était immense, mais les interactions entre participants de cultures différentes étaient limitées : les ingénieurs et les chercheurs qui bâtissaient le système entre 1977 et 1992 en étaient les principaux utilisateurs, c'était en quelque sorte un projet autarcique. Parler de projet est d'ailleurs quelque peu abusif, puisque personne n'avait imaginé son aboutissement ni son ampleur, et encore moins ses usages ! Le processus complexe d'autorégulation du développement qui s'est déroulé reste encore largement à élucider, ne serait-ce que sur le plan sociologique, mais le milieu social qui a construit l'Internet s'est construit lui-même en même temps et par le même processus. On pourrait risquer l'hypothèse de considérer l'Internet comme une *institution imaginaire* au sens où Cornelius Castoriadis emploie cette locution, qui désigne le fait que toute société humaine se construit elle-même, et que le moyen et le matériau de cette construction consistent en significations imaginaires (construites par l'imagination) auxquelles adhèrent les membres du corps social naissant. Bien sûr, le milieu social qui édifiait l'Internet au cours des années 1980 n'a rien à voir avec celui des consommateurs actuels du réseau.

La plupart des projets de la classe 3 correspondent peu ou prou au schéma que nous venons de décrire pour l'Internet : lors de la conception d'un système d'exploitation ou d'un compilateur, des informaticiens parlent à des informaticiens ; les utilisateurs finals n'interviennent pas, on ne voit d'ailleurs pas très bien comment ils le pourraient.

Il en va tout autrement pour les projets de la classe 1, qui édifient des systèmes d'information destinés à la gestion des entreprises. Les maîtres d'ouvrage sont le plus souvent dépourvus de culture technique, et le contexte est mobile – que ce soit pour de bonnes raisons ou par l'effet de la mode ne change pas grand chose aux conséquences de cet état de fait. Beaucoup de gens ont voix au chapitre, notamment, *a priori* et par définition, tous les secteurs de l'entreprise, avec des préoccupations diverses que chacun exprimera dans le contexte de son univers culturel propre : autant dire que l'établissement d'un langage commun ne sera pas facile, et la définition d'objectifs partagés pratiquement impossible. Un ouvrage récent [101] donne de ces difficultés une description satirique mais finalement assez réaliste.

Puisque la difficulté du projet de système d'information réside pour une part majeure dans la difficulté, éprouvée par des acteurs aux horizons culturels différents, à trouver un langage et des objectifs communs, la sagesse semble conseiller d'éviter dans la mesure du possible les interférences de ces acteurs entre eux, et avec le maître d'œuvre. Évidemment cette sagesse entre en conflit avec l'objectif même du système d'information, qui est d'accroître la cohérence de l'entreprise – mais le maximum de cohérence est-il toujours et partout le bon objectif ? Dans un établissement de recherche, par exemple, chaque laboratoire poursuit des activités autonomes, et la coordination peut sans doute se limiter à l'identification des

synergies utiles sur le plan scientifique, qui seront d'ailleurs souvent avec des laboratoires extérieurs à l'établissement, et à la création de services communs efficaces. La collecte et la consolidation des informations nécessaires à la bonne administration de la recherche n'imposent sans doute pas une centralisation générale. Beaucoup d'entreprises se trouveraient sans doute très bien d'une gestion assez décentralisée, et la mode des progiciels de gestion globale de l'entreprise, qui sévit depuis quelques années, n'est sans doute que l'effet d'une propagande efficace des fournisseurs auprès de dirigeants financiers qui y ont vu, non sans raison, un moyen d'étendre leur pouvoir : il semble bien aujourd'hui que le bilan du déploiement de ce genre d'outil se révèle assez peu favorable. Il serait sans doute sage de revenir à des solutions de gestion à centralisation modérée, qui n'imposent aux acteurs de l'entreprise que la cohérence nécessaire, déterminée avec doigté.

Il faut souligner ici que la complexité d'un système informatique de gestion n'est que la conséquence de la complexité de l'univers à gérer : il faut certes éviter que les logiciels n'accroissent arbitrairement celle-ci, mais il est vain d'espérer qu'ils puissent à eux seuls la réduire – la simplification éventuelle des procédures de gestion ne pourra résulter que d'une action sur l'organisation, rendue possible peut-être par l'informatisation, et à laquelle les logiciels contribueront. Toute informatisation doit être précédée d'une remise en cause de l'organisation et de la gestion au plus haut niveau, faute de quoi on aboutit au mieux à une préservation de l'existant qu'il aurait justement fallu modifier, au pire à un accroissement significatif et parfois fatal du désordre.

Il résulte de cette dernière proposition, comme nous l'avons déjà souligné, qu'il convient d'identifier clairement les rôles respectifs de la maîtrise d'ouvrage et de la maîtrise d'œuvre et d'assurer l'équilibre entre ces deux pôles. La disparition ou l'affaiblissement excessif de l'un ou de l'autre conduirait le projet à l'échec.

Une autre conclusion de nos observations relatives au rôle et à la place du système d'information, c'est que tout responsable d'entreprise devrait posséder un minimum de compétence en système d'information, et partant en informatique. Dans beaucoup de pays cela va de soi, mais pas dans le nôtre, semble-t-il. Il existe bien en France une « fracture numérique », mais elle n'est pas forcément où on le croit, ses victimes (qui en sont aussi les coupables) sont à chercher au sommet de la hiérarchie sociale, au sein des élites économique, culturelle et sociale, où l'on pense que toucher un clavier d'ordinateur, acte technique et partant subalterne, serait déroger à son rang.

Annexe A Pièges des contrats de logiciel

Sommaire

Au matin du 15 octobre 2015 j'ai répondu à l'invitation des sociétés Aspera et EasyTrust en assistant à la conférence SAM 2015 consacrée à la gestion des actifs logiciels, en compagnie de représentants des sociétés Carrefour, LVMH, Airbus Industries, Essilor, Natixis, Total, de la Direction générale de l'Aviation civile, etc. Ce fut passionnant.

A.1 Administrer un grand parc de logiciels est difficile

Il s'agissait de proposer des solutions à un problème que j'ai bien connu dans mes fonctions à l'Institut Pasteur, à l'Inserm et à l'Université Paris-Dauphine : l'administration des contrats de licence des logiciels commerciaux. Plus précisément, comment négocier et appliquer ces contrats en évitant deux écueils : violer certaines clauses, ce qui expose aux rigueurs de la loi, et se ruiner en payant plus cher que ce qui est dû.

Pour une entreprise la question ne se pose pas de la même façon que pour un particulier : il y a d'une part une instance qui conclut le contrat et en paye les redevances sur son budget, en général la Direction du Système d'information (DSI), et de l'autre des utilisateurs (des divisions ou des gens) qui ont tendance à se comporter comme des clients d'une ressource gratuite, ce qui rend la demande infinie. De l'autre côté, les éditeurs qui vendent les licences de droit d'usage des logiciels, ont tout intérêt à stimuler la fièvre acheteuse des utilisateurs finals et à inhiber les velléités régulatrices de la DSI ; nous allons voir qu'ils disposent pour ce faire de certains concours de circonstances et de dispositifs contractuels d'une grande habileté.

Les principaux éditeurs de logiciels dont la conférence SAM 2015 débattait sont Microsoft, Oracle, SAP, VMWare, IBM, Adobe. Les contrats types de ces éditeurs comportent en général des clauses qui autorisent le vendeur à mener chez le client un audit de conformité au contrat, c'est-à-dire une opération de vérification que le nombre de copies de logiciels déployées et les options activées correspondent bien à ce que prévoit le contrat et aux redevances versées.

A.2 Combien de logiciels dans mon parc ?

Il faut savoir que dans une organisation d'une certaine taille il n'est pas simple de savoir combien de copies de Microsoft Office ou d'Oracle Database sont déployées sur les postes de travail, les ordinateurs portables et les serveurs de l'entreprise, sur ses différents sites. En

effet la DSI dispose en général d'une ou plusieurs copies du logiciel, qui sont recopiées au fur et à mesure des besoins sur les différents matériels. Dès que l'organisation atteint une certaine taille la distribution des copies est décentralisée et du ressort d'informaticiens de terrain parfois éloignés du centre de gestion. En outre les logiciels sont souvent munis d'options multiples dont l'activation donne lieu à facturation supplémentaire, selon les termes de contrats de licence parfois difficiles à interpréter, à tel point que s'est développé une campagne d'opinion pour la clarification des licences[1].

Il est de la responsabilité du client de ne pas excéder le nombre de copies pour lesquelles il paye, ce qui est tout à fait légitime. Dans la pratique, les audits de conformité révèlent que la réalité observée excède souvent les termes du contrat. Il est rare que cette situation résulte d'une volonté de fraude délibérée. Le plus souvent, les utilisateurs finals ont tendance, lorsque l'informaticien vient leur installer le logiciel, à demander par précaution plus d'options que ce dont ils ont réellement besoin. Un gros client citait l'exemple d'un parc où l'utilisation d'un logiciel d'inventaire avait révélé la présence de 10 000 PC allumés, connectés au réseau et dotés de tous les logiciels usuels, mais sur lesquels aucun utilisateur ne s'était connecté depuis plus d'un an. Cette situation ouvrait droit à paiement de licences inutiles.

A.3 Conséquences de la virtualisation

Néanmoins, ce n'est pas sur les postes de travail que les plus gros débordements sont observés, mais dans les centres de calcul (*datacenters* comme il faut dire désormais). Il y a plusieurs raisons à cela, et toutes ne renvoient pas à la négligence des agents de la DSI. Un tel centre dispose en général d'une architecture redondante, tant pour les serveurs que pour les baies de stockage de données. Aujourd'hui la plupart des logiciels et des bases de données sont exploités sur des machines virtuelles, et parfois dans un nuage public *(cloud computing)*. Une propriété intéressante et utile des machines virtuelles est la facilité de leur multiplication au gré des variations dans le temps des besoins de calcul.

Le calcul de la base de facturation des droits d'usage dans un tel contexte na va pas sans prêter à débat. Les éditeurs ont tendance à interpréter la situation de telle sorte que dès lors qu'une copie du logiciel est activable sur une machine (réelle ou virtuelle) du centre, il faut payer pour toutes les machines de toutes les baies, parce que rien n'empêche le logiciel de s'y propager. Une telle attitude est de nature à inciter les clients à regrouper leurs applications en silos étanches pour échapper à la surfacturation, ce qui va à rebours des évolutions techniques et de l'intégration du système d'information pour une meilleure gestion de l'entreprise.

A.4 Comment payer le juste prix ?

Il y a deux moyens de rapprocher la réalité des clauses du contrat : réduire le déploiement réel de logiciels en supprimant les copies et les options inutiles ou superflues, ajuster les termes du contrat au besoin réel. Les entreprises organisatrices de la conférence SAM 2015, Aspera et EasyTrust, proposent des logiciels et des services qui permettent, d'abord, de mieux connaître la réalité du parc de logiciels déployés dans l'entreprise, d'autre part de mieux adapter les contrats aux besoins réels.

Lorsqu'un audit de conformité révèle un écart entre la réalité et le contrat au détriment de l'éditeur, cet écart donne lieu à un redressement par le paiement d'indemnités rétroactives. Il faut savoir que les revenus engendrés par ces redressements sont loin d'être marginaux.

1 Cf. http://www.clearlicensing.org/

Selon la communication du rédacteur en chef de la revue britannique *ITAM Review*[2], Martin Thompson, présent à la conférence, elles représentent pour la filiale britannique d'Oracle 50 % du chiffre d'affaires. Ces audits sont menés par l'entité Oracle LSM *(License Management Services)*, qui relève de la direction financière, et qui n'a donc aucune raison de faire au client une faveur commerciale. Dans le cas de Microsoft les audits sont confiés à des cabinets extérieurs rémunérés en fonction des sommes récupérées au titre des redressements. Bref, on voit qu'il vaut mieux être en règle lorsque se profile un audit de conformité.

A.5 Le paysage encombré et mouvant des licences

Comment se fait-il que clients et éditeurs puissent avoir des compréhensions très différentes du montant des redevances à payer pour le droit d'usage d'un logiciel ? Nous avons mentionné ci-dessus la possibilité d'interpréter de façon large le nombre de machines sujettes à redevance, ainsi que l'absence de clarté des termes de certaines licences. Ces questions peuvent être complexes, et en général le client ne possède pas toute la compétence nécessaire pour trouver le chemin optimal parmi la trentaine de formes différentes de licences que l'on observe, par exemple, chez Microsoft. Le vendeur, bien sûr, connaît cela sur le bout des doigts.

Pour compliquer encore la chose, les termes des licences évoluent dans le temps. Ainsi la licence standard édition 1 d'Oracle valait pour quatre *sockets*[3] sur un serveur, mais si le client veut passer à la version 12.1.02 du logiciel il passera *ipso facto* et sans forcément en avoir conscience sous le régime de la licence standard **édition 2**, qui ne vaut plus que pour deux sockets, du coup il ne sera plus en mesure de satisfaire un audit de conformité et il s'exposera à un redressement.

Les plans de reprise d'activité sont une autre source de confusion : une DSI prudente disposera pour ses systèmes vitaux d'un centre de secours, dans un de ses centres de calcul ou hébergé chez un prestataire extérieur. Ce centre de secours devra bien entendu être équipé de tous les logiciels nécessaires : faut-il payer les redevances plein tarif, sachant qu'en régime normal le logiciel n'est pas utilisé ? les systèmes installés chez l'hébergeur éventuel relèvent-ils des mêmes conditions contractuelles que ceux installés dans les locaux du client ? Autant de questions qu'il vaut mieux s'être posé avant l'audit de conformité.

A.6 Comment préparer un audit de conformité

La complexité de la situation exposée ci-dessus montre que l'entreprise qui aurait signé des contrats de droit d'usage de logiciels sans se donner les moyens de répondre à toutes les questions évoquées s'exposerait à des déconvenues financières qui peuvent se chiffrer assez vite en millions d'euros. Mon expérience à l'Inserm, entreprise de taille moyenne, m'a montré que cet ordre de grandeur n'était pas du tout irréaliste.

Comment éviter de telles difficultés ?

D'abord, il faut bien connaître son parc. Acquérir cette connaissance exige une démarche systématique au moyen d'outils automatiques, donc des logiciels spécialisés.

Cette démarche d'acquisition de la connaissance du parc installé peut être considérée comme une intrusion par certains utilisateurs. Il est donc indispensable qu'elle soit impulsée et soutenue au plus haut niveau de la hiérarchie : DSI, DAF, DG. Sinon elle échouera.

2 Cf. http://www.itassetmanagement.net/

3 Dans ce contexte, on nomme *socket* le socle sur lequel s'enfiche le processeur d'un ordinateur, il y a donc en général un processeur par *socket*, mais il y a des processeurs qui en occupent deux. Les processeurs modernes sont multi-cœurs : chaque cœur peut être envisagé comme un processeur autonome, ce qui ouvre la voie à de nouveaux débats sur l'assiette des licences.

Il faut aussi une connaissance fine des subtilités des contrats de licence et de leurs évolutions dans le temps et dans l'espace. Posséder de telles compétences n'est possible que pour de très grandes entreprises, et encore. Peut-être vaut-il mieux s'en remettre à des cabinets spécialisés.

Il faut aussi jouer avec le temps : si l'audit de conformité intervient à l'improviste sans que la DSI soit prête il a toutes les chances de donner un résultat catastrophique. Il est possible de différer l'audit, mais cette attitude a des limites, parce que les contrats obligent le client à s'y soumettre.

La conférence SAM 2015 a présenté deux expériences d'utilisateurs qui ont pu diminuer dans des proportions considérables leurs dépenses de droits d'usage de logiciels : la Direction générale de l'Aviation civile (DGAC) et *Deutsche Bundespost*-DHL (480 000 agents, 120 000 véhicules), ce au moyen d'une gestion rigoureuse et d'une bonne préparation des audits qui ont permis des négociations équitables avec les fournisseurs.

A.7 S'adapter au changement permanent des règles

Comme en 2015, les sociétés Easytrust[4] et Aspera[5] ont organisé une conférence fort instructive sur la gestion de parc logiciel, SAM ITAM 2016 *(Software Assets Management - IT Assets Management)*. Comme de juste l'assistance était surtout constituée de représentants de grandes sociétés (Société Générale, Orange, DGAC, Canal +, Harmonie Mutuelle, le Ministère de la Défense, Total, AXA...); la présentation d'expérience vécue de cette année était celle des Laboratoires Roche, 48 milliards de francs suisses de chiffre d'affaires[6], près de 100 000 salariés de par le monde, et qui sur une facture annuelle de l'ordre de 300 millions de francs suisses pour les logiciels a réussi à obtenir une réduction de plus de 60 millions par une meilleure gestion du parc, avec l'aide des sociétés organisatrices de la conférence et de leurs outils, que j'ai déjà décrits ci-dessus. Cet article précédent décrit également les procédures d'audit réalisées par les éditeurs chez leurs clients, et les clauses (léonines) de redressement en cas de dépassement de licence. Naturellement tout est fait pour savonner la pente sur laquelle le client naïf va glisser jusqu'au dépassement.

Nous avons eu un exposé d'Éléonore Varet, juriste chez Easytrust, qui nous a expliqué que les éditeurs de logiciels se faisaient parfois menaçants, mais qu'en fait ils n'aimaient pas trop la jurisprudence, qui ne leur était pas toujours favorable, et qu'une attitude ferme étayée par des arguments solides, fournis par exemple par de bons outils de *Software Assets Management*, pouvait atténuer leurs prétentions parfois abusives. Mais encore faut-il être un client suffisamment gros pour se faire entendre... Cet exposé était illustré de quelques exemples concrets de jurisprudences.

Easytrust nous a distribué des livres blancs aux titres alléchants et au contenu roboratif : *Les 10 choses à savoir en cas d'audit Oracle*, *L'optimisation des licences logicielles sur le terrain*, *Maîtriser et optimiser mes licences SAP*, *Gestion des logiciels Oracle : l'utilité d'une solution vérifiée par l'éditeur* (les logiciels Aspera sont vérifiés par Oracle).

Voici ce que je crois avoir compris de la politique de licence des éditeurs les plus retors. D'abord, les règles contractuelles, les définitions des notions utilisées pour calculer l'assiette des redevances et la façon de compter les objets changent en permanence. Ainsi, Microsoft propose une trentaine de modèles différents de licences, Oracle n'en propose que huit (dont certaines ne sont plus au catalogue mais continuent à vivre chez les anciens clients), mais assorties de 15 options et de 5 Packs. Ces licences peuvent être décomptées soit par utilisateur, soit par centre de données, soit par processeur. La notion de « processeur » peut

4 Cf. http://www.easytrust.com/

5 Cf. https://www.aspera.com/fr/

6 De nos jours le cours du franc suisse est proche de celui du dollar et un peu inférieur à celui de l'euro, la livre sterling devrait bientôt être rattrapée par ce peloton.

correspondre soit à une *socket*, soit à un *core*, soit à un *thread*, selon la version de licence ce n'est pas la même définition qui s'applique, et au fil du temps cela change. Évidemment, avec des machines virtuelles (VM) et un système de déploiement de VM à la demande comme OpenStack, les définitions deviennent byzantines et les calculs obscurs, pour ne pas dire arbitraires.

Une règle au doigt mouillé : en général les anciens contrats sont plus avantageux que les nouveaux, que votre commercial bien-aimé vient vous proposer. Mieux vaut garder les anciens.

L'idée générale (mais schématique) est la suivante : l'espace des paramètres qui déterminent le calcul de la redevance peut être représenté par un hyperplan dont les axes correspondent (par exemple) à :

— modèle de licence ;
— nombre de processeurs concernés (selon la définition en cours du processeur) ;
— effectif d'utilisateurs bénéficiaires ;
— options activées.

Pour chaque point de cet hyperplan (dans notre exemple 4 axes) on calcule la redevance et on place, selon un axe supplémentaire (dans notre exemple le cinquième), un point à la hauteur correspondante. On trace la surface qui passe par tous ces points et on choisit sur cette surface le point le plus bas, qui correspond à la redevance la plus faible.

Naturellement les utilisateurs naïfs (comme tel de mes anciens employeurs) se trouvent en des points de la surface beaucoup trop hauts, et payent trop cher. Les utilisateurs avisés (instruits par exemple par Easytrust et équipés des outils Aspera, ou de leurs confrères) s'arrangent pour cheminer vers les creux de la surface.

Quand trop de clients, devenus avisés, sont dans les creux, les éditeurs voient leurs revenus baisser ; ils modifient alors les règles, la surface se gondole différemment et ceux qui étaient nichés dans des creux se voient propulsés sur les cîmes. Il faut donc recommencer tous les calculs pour trouver le nouvel optimum. C'est un travail à plein temps.

Annexe B Le Projet Unicorn, un roman de Gene Kim

Une histoire de développeurs, de disruption et de survie à l'ère des data

Ce roman plein de rebondissements et d'affrontements à l'issue indécise se déroule au sein d'une entreprise américaine de vente au détail de pièces détachées et de produits de maintenance pour automobiles, *Parts Unlimited*, d'une valeur de quatre milliards de dollars. Les développeurs y défendent leur vie contre bureaucrates et financiers à courte vue, et pour cela ils doivent révolutionner l'entreprise de la cave au grenier. Mais les décisions des actionnaires sont sans appel !

L'auteur, Gene Kim, a exercé en tant que chercheur à l'Université d'Arizona (Tucson) et à l'Université Purdue, mais il a aussi participé à la création de trois entreprises, dans les secteurs DevOps et cyber-sécurité. C'est dire qu'il possède une solide expérience tant des aspects théoriques que de la vie quotidienne des applications informatiques. Livre publié aux Éditions Quanto[1] (Presses de l'École polytechnique fédérale de Lausanne).

B.1 L'incident déclencheur

Au premier chapitre, *Parts Unlimited* est victime d'un grave incident informatique : la paie de septembre est fausse, plusieurs milliers d'employés n'ont pas pu être payés ou ont reçu des montants erronés. La direction exige que tombent des têtes, et l'héroïne du roman, Maxine Chambers, cheffe développeuse chevronnée et talentueuse, est désignée pour porter le chapeau (avec quelques autres) : d'un poste opérationnel à haute responsabilité, elle est mutée à la documentation du projet *Phoenix*, « le projet maudit de l'entreprise depuis des années, tristement célèbre pour avoir empêtré des centaines de développeurs sans résultat. » L'objectif de *Phoenix* est l'intégration des services commerciaux de l'entreprise et le développement de la vente en ligne afin de résister aux nouvelles entreprises nées sur l'Internet et aux géants de la vente sur le Web, mais jusqu'à ce jour il n'en est sorti que des pannes de réseau, des serveurs engorgés et des bases de données corrompues.

B.2 Une entreprise mal gérée, comme beaucoup

En fait, le paysage informatique de *Parts Unlimited* sera familier à quiconque aura traîné ses guêtres quelques années dans cet univers :

- L'informatique n'est pas traitée comme un actif stratégique et un moteur de développement, mais comme un centre de coûts et un « service support ». - Quand les dirigeants trouvent que l'informatique coûte cher, ils l'externalisent, en totalité ou par parties. Puis ils constatent que les prestataires sont encore plus chers et assez mauvais, alors on réintègre ce qui avait été externalisé, avec de nouveaux personnels qui ne connaissent pas le terrain et commettent des bourdes. - Le travail des informaticiens est perçu comme celui de terrassiers : si cinquante terrassiers ne vont pas assez vite pour créer un remblai, on les fouette plus fort. Et si cela ne suffit pas, on en ajoute cinquante de plus. Mais l'informatique, qui mobilise le

1 https://www.epflpress.org/editeur/5/quanto

cerveau plus que les bras, ne fonctionne pas ainsi, ajouter à un projet des personnels qui ne le connaissent pas va surtout créer des incohérences, des pannes et des retards supplémentaires.
- Après quelques années de ces errements, le système d'information de l'entreprise est un vaste foutoir, pour corriger on empile des échelons hiérarchiques et des règles de *workflow* qui ne font qu'ajouter aux lenteurs et aux blocages, cependant qu'au sommet on se gargarise avec les mots à la mode (IA, *deep learning*, *Big Data*...) alors qu'il n'y a rien derrière.

B.3 Risque de désertion des actionnaires

Il suffit de quelques années de ces pratiques, assaisonnées d'impulsions narcissiques de dirigeants plus ou moins lucides, pour que les actionnaires se disent que leur argent serait mieux placé ailleurs, et envisagent la vente des meilleurs actifs et le rétrécissement du périmètre, voire la liquidation de l'entreprise.

Comme l'auteur est américain, il ne méprise pas les actionnaires, il sait que sans eux l'entreprise n'aurait pas les moyens d'investir à hauteur suffisante pour tenir face à la concurrence. À charge pour la direction de l'entreprise de montrer que leur confiance est bien placée.

B.4 Impuissance

En exil, donc, sur le projet *Phoenix*, Maxine se morfond parce qu'elle ne peut rien faire : pour accéder à la moindre information technique il faut passer par un système de tickets (pas toujours le même !) qui prend un temps fou, aux points clés il faut obtenir des autorisations hiérarchiques encore plus chronophages, et de toute façon, une fois obtenus les coups de tampon administratifs, les procédures qui lui permettraient d'assembler les différentes pièces de *Phoenix* sur son ordinateur (l'opération de *build*, leitmotiv de toute la première partie du roman) ne fonctionnent pas.

En effet, les multiples équipes de développement du projet, entassées sans rien y comprendre dans des *open spaces* par une direction qui croyait en la vertu des gros bataillons, et enfermées dans des silos hiérarchiques, ont travaillé chacune avec leurs méthodes et leurs outils, sans coordination les unes avec les autres, et pour tout arranger les meilleurs développeurs, qui avaient une connaissance orale de l'architecture générale, ont depuis longtemps quitté le projet, voire l'entreprise, sans mettre leur savoir par écrit. Les praticiens du SI reconnaîtront une situation familière.

B.5 Rébellion

Maxine va trouver une issue, qui lui sera indiquée par Kurt, responsable d'assurance qualité. Kurt a compris qu'il n'y avait rien à espérer des circuits bureaucratiques officiels, et il organise la circulation clandestine des informations utiles. Il remet à Maxine un épais dossier qui contient les clés de licence, les identifiants pour accéder aux environnements de développement et tout ce qu'il faut pour générer un *build* de *Phoenix*. Munie de ces informations indispensables, Maxine va pouvoir construire une instance de *Phoenix*, commencer à travailler et, ce faisant, commencer à débusquer les incohérences et les goulots d'étranglement du projet, bref, donner la mesure de ses talents de programmeuse.

En fait, Kurt anime un petit groupe clandestin au sein de l'entreprise, la *Rébellion*, qui réunit des personnalités diverses de *Parts Unlimited*, mais surtout des « militants de base », des développeurs, des responsables d'exploitation ou de sécurité opérationnelle, tous décidés à sortir *Parts Unlimited* de sa léthargie en court-circuitant les procédures officielles. Maxine va bien sûr se joindre à eux.

Leurs adversaires ? Les bureaucrates confits dans l'immobilisme et la hiérarchie, et surtout, plus dangereux, les membres de l'état-major qui pensent que le meilleur moyen de s'attirer les bonnes grâces des actionnaires serait de vendre des actifs, de licencier le personnel correspondant, et ainsi de leur distribuer de l'argent. Ce serait une stratégie de décroissance de *Parts Unlimited*, considérée comme radicalement obsolète. La Rébellion veut relever ce défi, ce sera une lutte d'appareils.

B.6 Le redressement ou la mort !

Maxine et ses nouveaux amis vont entreprendre leur travail de résurrection du système d'information de l'entreprise en commençant par les composants les plus pourris. Maxine est une adepte de la programmation fonctionnelle, familière du langage Lisp : en appliquant ces principes, on arrive à découpler les différents composants du système, ce qui permet de les corriger et de les améliorer sans interférer avec les autres composants. En procédant ainsi les membres de la Rébellion vont réussir à dénouer petit à petit l'immense et inextricable plat de spaghettis en quoi consiste le SI de *Parts Unlimited*. Mais l'ennemi ne renonce pas à ses plans néfastes : je ne divulgâche pas la fin...

B.7 Un roman pour les développeurs

Bon, l'intrigue de ce roman se déroule dans un univers de développeurs, il emploie leur jargon et parle des objets de leur travail quotidien, que le commun des mortels ne connaît pas. Si le notion de *build* ne vous dit rien, si un passage tel que celui-ci : « ne perds pas de temps à essayer d'accepter des noms de fichiers contenant des espaces dans tes Makefiles, c'est trop compliqué. Impose l'utilisation d'un autre caractère. » ou celui-là : « Il y a dix ans, se souvint-il, j'ai perdu mon fichier de configuration Emacs et je n'avais pas de sauvegarde récente. Je n'ai pas eu le courage de tout reprendre à partir de zéro... alors j'ai simplement changé d'éditeur ! » n'évoquent rien pour vous, il va falloir sauter pas mal de passages (entre parenthèses, c'est très mal d'abandonner Emacs).

Mais si vous faites l'effort de sauter par dessus ces obstacles, ou si vous êtes tout simplement de la partie, ce roman vous plongera au cœur des problèmes de l'industrie contemporaine. Parce que pour les entreprises traditionnelles, il n'y a pas trente six alternatives si elles veulent survivre face aux entreprises créées dès le départ autour de leur SI, telle Amazon : soit elles s'imposent les transformations qu'imposent Maxine et ses petits camarades à *Parts Unlimited*, soit elles disparaissent.

Annexe C À l'origine de la raison critique

Sommaire

C.1 Pour comprendre la nature du travail de conception

Les chapitres de ce livre ont traité de certaines activités des hommes, et notamment du travail, plus précisément d'un travail de conception qui par définition implique la pensée. Il me semble impossible d'aborder de tels sujets sans préciser quelque peu ce que cela veut dire, la pensée et le travail, et comment cela se passe, l'homme au travail dans la société, ce qui fera l'objet de cette annexe, et aussi comment (à mon avis et très schématiquement) on peut se représenter l'activité économique contemporaine.

Nous allons donc, le plus brièvement possible, évoquer l'organisation des hommes en sociétés, familles, entreprises, classes et autres collectivités, sans oublier les groupes d'appartenance linguistiques, culturels ou religieux. Pourquoi est-ce indispensable ? Prenons un exemple : si une entreprise cherche un site pour installer ses équipes de développement de logiciel, tout le monde sent bien que cela ne vas pas du tout se passer de la même façon selon qu'il choisira Paris-La Défense, Casablanca, Bangalore ou Kyïv. De même, il est manifeste que le type de logiciel qu'il va pouvoir réaliser et les méthodes qu'il devra utiliser pour ce faire ne seront pas les mêmes selon qu'il recrute des ingénieurs avec un bagage scientifique ou des programmeurs formés sur le tas et orientés vers la gestion. Mais en rester sur ces impressions vagues n'est pas satisfaisant. Il existe des façons systématiques d'aborder de telles questions, ce sont des disciplines scientifiques qui s'appelllent la sociologie, l'anthropologie, l'économie politique et la géographie humaine. Alors, sans en devenir des spécialistes, il est bon d'avoir une idée du type de problèmes auxquels elles peuvent nous aider à réfléchir, et d'essayer ainsi de les aborder de manière rationnelle, plutôt que d'en rester à la sagesse populaire, toujours véhicule de préjugés en ces affaires. Bref, il s'agit de mieux comprendre ce qui se trame lorsqu'une entreprise demande à une équipe d'informaticiens (de la maison ou d'une société de services) de construire un pan de son Système d'information. Si tout cela se passait habituellement sans encombre, nous pourrions nous dire que ce détour est inutile, mais tous les professionnels de la chose savent que cela se passe souvent très mal, alors il faut réfléchir.

En arrière-plan de notre propos sur le Système d'information et la démarche qui permet (ou pas) de le réaliser, se jouent les questions du lien social, de la norme sociale, du pouvoir et de l'autorité, du langage et de la connaissance, enfin du travail et de la création. Il ne saurait bien sûr être question de traiter ici en détail ces immenses sujets, mais il convient de les situer sur notre terrain d'investigation, puisqu'aussi bien les aberrations et les échecs que nous décrirons résultent généralement de leur ignorance quand ce n'est pas de leur dénégation.

Plusieurs disciplines scientifiques leur sont dévolues et nous essaierons de tirer parti de leurs résultats. Après tout, si les chercheurs travaillent, c'est pour que les praticiens tiennent compte de leurs travaux.

Aussi, sans avoir la prétention d'en résumer la teneur, convient-il de dresser ici un tableau de leur territoire pour avoir une idée de ce que nous pouvons en attendre. Nous n'aurons garde ce faisant d'oublier le caractère mouvant et assez conventionnel des frontières selon lesquelles se partagent entre ces différentes disciplines les thèmes de l'étude des groupes humains.

En fait, cette brève évocation de domaines de recherche a aussi pour but de rappeler qu'un certain nombre de pratiques et de formes sociales, présentes de façon générale dans les sociétés occidentales contemporaines, et dont nous avons de ce fait souvent l'illusion qu'elles sont inhérentes à toute société humaine, sont en réalité historiquement datées, culturellement situées, bref contingentes et pas universelles. C'est important, parce que les collaborations basées sur des sous-entendus implicites sont lourdes d'échecs dès lors que les dits sous-entendus se révèlent n'être pas les mêmes pour toutes les parties. Bref, une approche multi-culturelle est nécessaire pour construire le Système d'information (SI), parce que même si tous les participants sont citoyens d'un même pays, ils ont (par définition de ce qu'est le SI) des formations, des origines sociales et des objectifs professionnels variés. Ceci est bien sûr encore plus vrai pour un projet international.

La sociologie :

La sociologie étudie les organisations sociales, les relations entre les groupes humains actuels ou du moins récents : classes sociales, catégories socio-professionnelles, associations, groupes de sociabilité, familles... ainsi que les attitudes et les stratégies des individus au sein de ou face à ces groupes. L'échelle de temps des phénomènes étudiés par la sociologie va de la décennie au siècle, et embrasse rarement des périodes pluri-séculaires, qui sont plutôt du domaine de l'histoire sociale. Ainsi, le sociologue de la famille se penchera par exemple sur les relations entre les générations au sein du groupe familial, qui évoluent à un rythme assez rapide, spécialement depuis un siècle, alors que les structures de parenté, phénomène beaucoup plus stable, relèvent plutôt de l'anthropologie.

C.2 Le langage, facteur de cohésion *et* de désagrégation sociale

Un trait important de l'humanité est son usage d'un langage élaboré. L'importance du langage pour comprendre tout phénomène humain, et en particulier les difficultés de conception d'un système d'information, n'échappera à personne. Il y a peu prévalait encore l'idée que le langage était l'apanage exclusif de l'homme. Encore plus récemment son apparition était datée de l'apparition d'*Homo sapiens*, il y a à peu près 100 000 ans. Les paléo-anthropologues envisagent maintenant des hypothèses d'apparition beaucoup plus précoce du langage. Les primatologues et les éthologues ne cessent de mettre en évidence des compétences langagières de plus en plus élevées chez les singes et chez d'autres mammifères,

pratiques qui dépassent largement les signaux qu'échangent les insectes collectifs et d'autres animaux. La question de l'hominisation, empêtrée dans la biologie, cède alors la place à celle de l'anthropogenèse.

L'acquisition par l'homme du langage et du pouvoir considérable qu'il confère par la combinaison infinie des symboles et les innovations qui peuvent en découler n'allait pas sans danger.

Anthropologie (1) :

La branche principale de l'anthropologie jusque dans les années 1930 était l'anthropologie physique, qui étudiait comme son nom l'indique les caractères physiques de groupes humains que l'on appelait à l'époque des races. Certains événements historiques ont jeté un violent discrédit sur cette discipline aujourd'hui éteinte (au moins dans les milieux scientifiques respectables) et l'anthropologie se consacre aujourd'hui à l'étude de structures sociales plus « profondes » et moins mobiles que celles auxquelles s'intéresse la sociologie : structures de parenté, faits religieux, terrain où elle rencontre éventuellement l'ethnologie. Il existe bien entendu de multiples champs partagés entre sociologie et anthropologie, qui s'y distinguent par leur angle de visée. Ainsi la sociologie religieuse s'intéressera plutôt aux pratiques et à leur variabilité, cependant que l'anthropologie religieuse mettra en lumière les éléments permanents au sein d'un groupe, les mythes fondateurs, et ce qui les rattache aux structures de parenté, par exemple.

Dans les conditions de la vie préhistorique, la cohésion sans faille des groupes humains était un impératif vital. L'homme est un animal social dont chaque comportement est dirigé par le besoin de l'autre : l'irruption du langage mettait en danger ce lien social et rendait nécessaire l'invention de moyens nouveaux, langagiers, en fait, pour assurer que chaque membre du groupe utilise les mêmes combinaisons de signes pour exprimer les mêmes réactions aux mêmes phénomènes, et que l'élaboration symbolique ne crée pas de conflits ou de scissions, bref assurer l'unanimité du groupe autour de symboles communs combinés selon des conventions partagées. C'est pourquoi le conformisme, qui nous apparaît aujourd'hui plutôt comme un défaut, a été pendant la plus grande part de l'histoire de l'humanité une règle de comportement vitale.

C.3 Naissance de la démarche critique

L'aspiration à l'unanimité, la répression des élaborations symboliques dissidentes ou simplement originales sont encore très présentes dans l'humanité contemporaine, et chacun en observera les effets autour de lui.

Ce livre envisage des activités humaines où le conformisme et l'unanimité pèsent souvent plus lourd que l'originalité et la liberté. Néanmoins, si les sociétés européennes ont connu depuis le Moyen-Âge un développement inconnu des autres régions du monde, c'est parce qu'elles ont remis en cause ces attitudes. Dans les sociétés traditionnelles, l'idéal est l'immobilisme et la conservation des traditions, ce qui fait que les évolutions ne pourront y être que très lentes, contingentes et souvent provoquées par des événements extérieurs, alors que dans les sociétés qui ne se soumettent plus à cette règle le changement et l'innovation sont des valeurs positives et encouragées, ce qui suscite des capacités d'adaptation et d'expansion supérieures. Ces questions ont été étudiées de façon pénétrante par Cornelius Castoriadis [26]. La question de l'acceptation d'un point de vue différent chez autrui n'est pas seulement collective et sociale, mais aussi psychologique et individuelle, ce qui n'est pas sans répercussion dans les processus liés au travail que nous allons envisager ici. Accepter qu'autrui puisse avoir un point de vue différent du vôtre et néanmoins légitime sur un sujet qui vous concerne comme lui, admettre la nécessité d'une négociation et de transactions avec ce point de vue que vous n'approuvez pourtant pas, sont des attitudes généralement

considérées comme caractéristiques de l'âge adulte, par opposition aux comportements observés chez les adolescents.

Avec la prolongation remarquable de l'adolescence jusqu'à trente ans, voire au-delà, signalée par les psychologues dans les pays occidentaux, il est devenu imprudent de postuler une attitude adulte au sein du personnel (même hautement qualifié) d'une entreprise.

De toutes les façons, parler du travail en entreprise aujourd'hui serait impossible en ignorant la question de la résolution des conflits entre points de vue différents, dans la mesure où l'ancien système d'autorité hiérarchique est fortement ébranlé.

C.4 L'écriture, la liberté

Nous avons déjà noté que les capacités d'expression infinies du langage humain conféraient à l'homme des pouvoirs nouveaux tout en introduisant dans la vie de ses sociétés des risques engendrés par cette créativité même, susceptible d'être un germe de conflits. C'est le prix à payer pour une pensée protéiforme.

Est-il besoin de souligner combien l'invention de l'écriture a multiplié le facteur de diversité que représente le langage ? Les sociétés de culture orale transmettent leurs traditions par la récitation, le chant épique, l'art des bardes. L'écriture fournit un moyen technique de fixation de la mémoire, mais permet aussi la distance critique. Le pédagogue contemporain Philippe Meirieu et le journaliste Marc Guiraud ont pu écrire [81] : « Lire et écrire, c'est [...] prendre du pouvoir sur sa propre vie... Savoir lire, c'est pouvoir vérifier, comparer, critiquer... Écrire, c'est pouvoir exprimer ce que l'on pense et ce que l'on croit, prendre le temps de se corriger, de trouver la formule juste en évitant de s'exposer à la réaction immédiate de l'autre » (p. 12). Il est patent qu'entre l'époque où le lien de l'autorité et le lien religieux se mettaient en place pour assurer la cohésion sociale, et aujourd'hui où donner au jeune les moyens de prendre ses distances par rapport à la norme est un des objectifs proposés au système éducatif, une transformation a eu lieu : c'est la naissance de l'individu.

Parmi les langues écrites, l'écriture alphabétique phonétique inventée par les Grecs a représenté plus qu'un simple progrès technique. Une écriture idéographique comme celle du chinois demande au lecteur de connaître par cœur les caractères du texte qu'il lit. Une écriture alphabétique non vocalisée, comme celles de l'arabe ou de l'hébreu, ne permet pas de lire correctement un mot que l'on ne connaît pas au préalable ; il faut en référer à une autorité qui tranchera les ambiguïtés, dira en vertu de son interprétation du contexte de quel mot il s'agit et délivrera le sens du texte. La fixation, évoquée ci-dessus, de l'interprétation orthodoxe du Coran a consisté pour une part substantielle à déterminer une vocalisation canonique du texte. En ce qui concerne l'hébreu, on trouvera une analyse détaillée des difficultés d'interprétation du texte biblique au chapitre VII du Traité théologico-politique de Spinoza, qui confirme, et bien au-delà, les limites des alphabets non vocalisés ; ce texte est proprement renversant de modernité. L'alphabet vocalisé sape l'argument d'autorité en permettant à chacun d'aborder plus facilement des textes nouveaux et de les interpréter librement. La pratique d'une telle écriture favorise l'étude de questions inédites, pour lesquelles il faut inventer des réponses nouvelles, et pour ce faire adopter une démarche qui relève de la pensée spéculative. C'est justement dans ce domaine de la pensée spéculative que l'apport de la civilisation grecque a été prédominant.

C.5 Démocratisation de l'écriture

Mais l'invention grecque de l'écriture phonétique fut libératrice par d'autres aspects. Dans son petit livre *Les origines de la pensée grecque* Jean-Pierre Vernant [114] évoque la destruction de la civilisation mycénienne par l'invasion dorienne au XIIe siècle avant notre ère, et la disparition de l'écriture en Grèce qui en résulta. « Quand les Grecs la redécouvriront,

vers la fin du IXe siècle, en l'empruntant cette fois aux Phéniciens, ce ne sera pas seulement une écriture d'un type différent, phonétique, mais un fait de civilisation radicalement autre : non plus la spécialité d'une classe de scribes, mais l'élément d'une culture commune. Sa signification sociale et psychologique se sera aussi transformée — on pourrait dire inversée : l'écriture n'aura plus pour objet de constituer à l'usage du roi des archives dans le secret d'un palais ; elle répondra désormais à une fonction de publicité ; elle va permettre de divulguer, de placer également sous le regard de tous, les divers aspects de la vie sociale et politique » (pp. 31-32).

C.6 Langage, pensée, connaissance

C'est sans doute à la philosophie qu'il faut demander ce qui fait l'originalité de l'usage humain du langage. Ludwig Wittgenstein [132] et Moritz Schlick [100] me semblent avoir dégagé de façon décisive cette question de l'ornière spiritualiste tout en évitant le fossé du positivisme vulgaire.

Notre langage (j'évite ici la locution « langage naturel », bien qu'elle soit due à Bertrand Russel, parce qu'elle me semble de nature à introduire une ambiguïté entre la nature humaine et sociale, qui est bien la place du langage, et la nature au sens biologique) est formé de signes qui dans certaines conditions d'énonciation sont signifiants.

Moritz Schlick [100] fut de la fin de la guerre de 14-18 jusqu'à son assassinat par un nazi en 1936 le chef de file du cercle de Vienne, groupe de philosophes qui se détournaient de la métaphysique (ou en tout cas de certaines de ses préoccupations) et qui ancraient leur réflexion dans une véritable pratique scientifique. Schlick, physicien de formation, était titulaire de la chaire de philosophie des sciences à l'université de Vienne, et il a par exemple écrit un texte intitulé *La Signification philosophique du principe de la relativité* qu'Albert Einstein lui-même considérait comme une contribution notable à la pensée scientifique. Pour lui « si le rôle de la science [était] la découverte de la vérité, celui de la philosophie [était] la découverte du sens ».

Dans le langage Schlick envisage d'une part des signes, associés dans certaines conditions, comme pour Saussure, à des signifiés, d'autre part des formes d'expression qui combinent des signifiants de façon créative et donnent par là au langage humain son pouvoir infini de décrire des objets nouveaux et d'exprimer des idées nouvelles, ce qui est peut-être un caractère exclusif de l'humanité. Des travaux récents suggèrent que cette capacité d'expression illimitée du langage humain, inaccessible à d'autres animaux, serait liée à la structure récursive des grammaires de nos langues : une locution de la forme « si..., alors... » nous est compréhensible quels que soient le nombre de mots et leur agencement syntaxique entre « si » et « alors », tandis que des grammaires plus simples, comme celle qui ne serait dotée que de structures telles que « sujet — verbe — complément d'objet », ne procurent pas les mêmes possibilités d'extensibilité.

Pour Schlick la connaissance est en quelque sorte un fait de langage : tout ce qui est connaissable est exprimable, et ce que l'on peut en dire constitue la totalité de ce que l'on en connaît. Cette proposition tient compte bien sûr de l'extensibilité infinie du langage humain, mentionnée à l'alinéa précédent, qui permet l'apparition de notions et de connaissances nouvelles. Il ne saurait y avoir de « connaissance ineffable », cette locution même est un non-sens[1].

Schlick s'est plus particulièrement préoccupé d'un certain type d'expressions, les propositions logiques, c'est-à-dire les phrases dont on peut déterminer si elles sont vraies ou fausses. Le sens d'une proposition est contenu dans l'énoncé de la procédure qui permet

1 Précisons que pour soutenir cette définition, Schlick prend soin d'écarter l'usage du mot connaissance pour évoquer l'intuition, ou la « conscience immédiate », qui nous permet par exemple de reconnaître les visages alors qu'il nous est impossible de décrire un visage avec des mots.

de déduire sa vérité ou sa fausseté, ou en d'autres termes, pour emprunter le langage de la logique, sa valeur de vérité.

Cette procédure de vérification peut comporter des opérations formelles sur les termes de la proposition, comme le remplacement d'un terme par un autre ou par une locution plus longue, et des expériences qui permettront également de transformer la proposition selon leur résultat. Ces transformations, ces opérations sur les termes de la proposition, que l'on appelle des « réductions », aboutissent à des termes irréductibles. Ainsi, je ne sais rien dire de la couleur verte, ou de la peur : seule la vision de la couleur verte, ou le fait d'avoir éprouvé la peur, peuvent me donner accès au contenu de ces termes, au sujet duquel je ne peux, par ailleurs, énoncer aucune proposition. Il en va de même pour le daltonien ou l'aveugle, qui ont de la couleur verte une appréhension particulière, mais de même nature logique que celle de la personne dotée de la vision habituelle. Les énoncés logiques possibles qui concernent la couleur verte sont limités : je peux dire « c'est vert » au sujet d'un objet, et cet énoncé aura la valeur vrai ou faux selon que l'objet est, en fait, vert ou non.

Il n'y a pour Schlick aucune autre forme de connaissance que celle que nous venons de présenter sommairement, et les milliers de pages que la métaphysique a consacrées à la connaissance de réalités transcendantes sont nulles et non avenues. Comme Wittgenstein, dont il fut proche, il conclut que, à propos de ce dont on ne peut rien dire, il convient de se taire.

C.7 Langage et lien social

Pour résumer en quelques mots ce qui demanderait en fait des volumes, la création de pratiques symboliques partagées et unanimes est communément appelée religion. L'étymologie de ce terme est d'ailleurs parlante : ce qui relie (les hommes entre eux). Il est également permis de supposer que l'art est apparu pour des raisons similaires à celles qui ont engendré la religion, dont il était d'ailleurs probablement essentiellement une expression.

Le philosophe René Girard [49], poursuivant la pensée de certains anthropologues, a formulé quelques hypothèses fortes sur la nature originelle de la religion, pour lui indissociable de l'anthropogenèse. Le groupe humain menacé d'éclatement par les conflits symboliques conjurerait le danger en en imputant la responsabilité à un de ses membres, élu au rôle de victime émissaire et sacrifié. Rappelons ici la définition fameuse de Jean Hubert et Marcel Mauss : « Le sacrifice établit une communication entre le monde sacré et le monde profane par l'intermédiaire d'une victime », victime en l'occurrence d'ailleurs bien souvent mangée. Il semble en effet que toutes les sociétés primitives aient pratiqué le sacrifice humain et l'anthropophagie, dont les traces surgissent, dès qu'on se donne la peine de les chercher, y compris – sinon surtout – dans les manifestations les plus élevées et les plus raffinées de nos cultures.

Les impératifs nouveaux de la production et de la normalisation symboliques ont sans doute joué un rôle dans la naissance de la division du travail et de la hiérarchie au sein de la société. Le groupe primitif a sans doute eu assez vite besoin de spécialistes pour interpréter les signes surnaturels et donner un sens au monde. Les précurseurs de l'anthropologie moderne, tel par exemple Maurice Leenhardt [72], ont montré que dans l'univers de l'homme primitif chaque objet était doté d'un sens précis qui contribuait au sens global du monde, et que le devoir de l'homme était de préserver cet ordre global et d'y contribuer par des rites et par le respect d'interdictions ou d'obligations. C'est le sens des multiples interdits et tabous des religions primitives. Singulièrement, Maurice Leenhardt au contact des Canaques retrouve le sens des livres de l'Ancien Testament, qu'il avait étudiés pendant ses années à la Faculté de Théologie Protestante de Montauban : « Quel admirable code de la société primitive que le Lévitique. Je n'ai encore rien trouvé chez les Canaques que je ne retrouve chez les Hébreux et dans beaucoup de peuples sur lesquels j'ai pu lire. » La religion de la Torah, en effet, est

de ce point de vue encore très liée aux origines primitives de la religion, cependant que le christianisme, par son rejet des interdits alimentaires et des marquages symboliques tels que la circoncision, et par son universalisme, fonde une religiosité radicalement moderne et dégagée de la préhistoire.

Le psychiatre Daniel Marcelli [79], lui aussi sur une piste ouverte par des anthropologues, étudie un phénomène qu'il appelle humanisation, et qui n'est sans doute pas éloigné de l'anthropogenèse. L'humanisation serait indissociable de l'autorité, nécessaire notamment aux parents pour accomplir leur mission de protection et d'éducation des enfants. Mais l'autorité trouverait son origine dans les situations où l'homme préhistorique, pour compenser sa faiblesse physiologique individuelle, aurait dû élaborer des liens de connivence et de confiance avec ses congénères, par exemple pour chasser de gros animaux ou se défendre contre des agresseurs. Marcelli pense que c'est ainsi que l'homme aurait adopté un comportement inconnu des autres espèces animales : l'échange de regards pour se concerter, agir de concert. De cette pratique partagée serait née l'autorité d'un chef nécessaire à la coordination de l'action pour la chasse ou la guerre, autorité dévolue non pas, comme chez certaines espèces animales, à l'individu le plus fort ou détenteur de certaines caractéristiques physiques, mais à celui qui se serait révélé le plus apte à recueillir la confiance du groupe pour l'action.

Anthropologie (2) :

Une digression qui ne nous éloignera qu'en apparence de notre sujet, et qui nous fera toucher du doigt la nécessité de se méfier des idées toutes faites, nous mène à parler ici des structures de parenté, qui constituent sans doute l'objet central de l'anthropologie contemporaine. Loin d'être une notion purement biologique, elles sont une élaboration sociale ; il ne s'agit pas seulement de savoir qui sont le père et la mère physiologiques d'un membre de l'espèce humaine, mais de placer chaque membre du groupe à sa place dans un réseau symbolique chargé de sens et d'en tirer des conséquences pratiques : qui habite avec qui, qui doit le respect à qui, qui nourrit qui, etc. N'oublions pas d'ailleurs que jusqu'à une époque récente, si la maternité était un fait, la paternité était un attribut imaginaire, décerné à un homme avant même la conception de l'enfant, en vertu de règles de droit : la cérémonie du mariage décidait, entre autres choses, que désormais l'époux serait le père des enfants de cette femme qu'il épousait. Jusqu'en 1977 la loi française n'a pas dit autre chose. Hervé Le Bras et Emmanuel Todd [22] ont montré que le territoire d'un pays apparemment assez homogène tel que la France abritait en fait plusieurs modèles anthropologiques qui influaient fortement sur la tonalité sociale, politique et psychologique de leurs aires respectives d'extension.

Nous ne ferons que citer le processus qui d'un clan doté d'un chef et d'un chamane conduit à une société plus complexe dotée d'organes de pouvoir et de structures religieuses spécialisés. De même que la division du travail s'est traduite par l'apparition de spécialistes du symbolique, elle a aussi engendré des professionnels du pouvoir politique. Le pouvoir est sans doute une fonction indispensable à toute organisation humaine, pour parler en son nom il lui faut quelqu'un qui s'en détache et se place pour ainsi dire en dehors d'elle. Toute une part de la complexité des questions soulevées par l'existence du pouvoir tient à l'impossibilité de séparer d'une part le caractère nécessaire de l'exercice de la fonction, d'autre part le fait indubitable que celui qui l'exerce en tire des avantages personnels substantiels, complexité encore accrue par les liens étroits et inextricables qui existent entre le pouvoir et le sacré.

C.8 Raison critique et autonomie

Plusieurs sociétés antiques ont connu des périodes d'assouplissement de l'exigence unanimiste pendant lesquelles sont apparues des formes de ce que l'on peut appeler la

pensée critique : la Grèce à l'âge classique (mais que l'on se rappelle le sort de Socrate !) et hellénistique, Rome sous l'Empire, le monde arabo-musulman jusqu'au XIIIe siècle avec notamment le courant philosophique mutazilite qui prône une pensée musulmane rationaliste (on lira à ce sujet avec profit les livres du regretté Mohamed Arkoun, qui pour les profanes a même eu l'amabilité d'écrire un excellent petit « Que sais-je ? » [7]). Les écrits de Confucius ne laissent aucun doute sur l'intensité du conformisme chinois antique, mais prouvent par leur existence même qu'il existait des espaces d'expression libre.

Cependant la naissance d'une pensée libre de remettre en cause les croyances communes fut sans doute un événement unique qui eut lieu en Grèce. Cornelius Castoriadis [27] écrit notamment : « La philosophie naît, en Grèce, simultanément et consubstantiellement avec le mouvement politique explicite, démocratique. Les deux émergent comme mises en question de l'imaginaire social institué…

La question "Pourquoi notre tradition est-elle vraie et bonne ? Pourquoi le pouvoir du Grand Roi est-il sacré ?" non seulement ne surgit pas dans une société archaïque ou traditionnelle, mais surtout elle ne peut pas y surgir, elle n'y a pas de sens. La Grèce fait exister, crée, ex nihilo, cette question. La représentation, l'image socialement établie du monde n'est pas le monde. Ce n'est pas simplement que ce qui apparaît diffère, banalement, de ce qui est ; cela, tous les primitifs le savent – comme ils savent aussi que les opinions diffèrent de la vérité. »

Toutes les sociétés se sont créées elles-mêmes en créant leur lois, mais Castoriadis distingue les sociétés hétéronomes, où l'origine des lois est attribuée à une instance extérieure (Dieu, les Ancêtres…), des sociétés à tendance autonome, où existe la conscience de cette création des lois par la société elle-même, et donc du pouvoir que possède la société de modifier les lois. Dans une société hétéronome, pour laquelle par exemple les lois sont divines, il est bien sûr hors de question que les hommes puissent les modifier ou les contester, ce qui a pour conséquence accessoire que l'idée même de démocratie politique[2] y est parfaitement absurde. L'autonomie est une tendance plus ou moins prononcée de telle ou telle société, on ne peut pas parler de société autonome – simplement, dans toutes les sociétés, il y a des individus autonomes, qui constituent des ferments d'évolution ; d'autre part certaines sociétés inscrivent consciemment et explicitement le droit à l'autonomie dans les institutions et dans les faits, et il sera possible alors de parler de société à tendance autonome.

Je ne voudrais pas suggérer ici qu'il y aurait un progrès social, de l'hétéronomie vers l'autonomie, parce qu'une telle notion serait très problématique, ne serait-ce que parce que certaines des tyrannies les plus terribles qu'ait endurées l'humanité avaient fait table rase de l'hétéronomie, et avaient même tiré parti de la liquidation des anciennes valeurs religieuses et morales pour déchaîner des exactions terrifiantes. Il faut souligner aussi que les sociétés qui revendiquent leur autonomie, comme par exemple la nôtre, ont développé des procédés de formatage et d'uniformisation des individus qui engendrent un conformisme souvent bien plus profond que ce que l'on peut observer dans certaines sociétés en principe hétéronomes. Ce que l'on peut néanmoins noter avec C. Castoriadis, c'est que dans une société hétéronome l'idéal est l'immobilisme et la conservation des traditions, et que les évolutions ne pourront y être que très lentes, contigentes et souvent provoquées par des événements extérieurs, alors que dans une société à tendance autonome le changement et l'innovation sont des valeurs positives et encouragées, ce qui suscite des capacités d'adaptation et d'expansion supérieures.

Le christianisme a établi sur les décombres de l'Empire romain d'Occident une hétéronomie et une exigence d'unanimité qui furent ébranlées d'abord au XIe siècle par la création des premières communes bourgeoises, puis par l'irruption au XIIe siècle de la philosophie grecque (essentiellement celle d'Aristote) redécouverte et introduite en Occident par les

2 Il convient de distinguer la démocratie sociale, qui consiste en ce qu'aucune fonction sociale n'est réservée à une catégorie particulière de personnes distinguées par leur naissance, de la démocratie politique, qui établit le pouvoir du peuple, c'est-à-dire le fait que le peuple crée les lois.

Arabes, alors que le christianisme antérieur avait plutôt puisé chez Platon et Épicure dont les doctrines étaient nettement plus compatibles avec le dogme. Il est d'ailleurs plausible que la survie en Occident de manuscrits de Platon et d'Épicure et la disparition de ceux d'auteurs moins faciles à adapter à la doctrine chrétienne soient directement liées à cette compatibilité. Il n'est pas besoin de recourir ici à l'hypothèse d'un autodafé de manuscrits subversifs par les moines médiévaux : ceux-ci ont simplement conservé précieusement les manuscrits qui leur semblaient de grand intérêt, quant aux autres ils les ont grattés pour réutiliser le parchemin ; l'examen des palimpsestes nous apprend quels auteurs furent « grattés », quand et pour être recouverts par qui. Reste qu'en « grattant » Aristote ces moines avaient vu juste, parce que sa réapparition allait contribuer à une évolution qui conduirait, *in fine*, à la sécularisation de la société. Le foisonnement de la pensée théologique scolastique qui résulta de ce retour d'Aristote, d'abord en Italie avec la création des premières universités (Bologne fondée en 1088, Padoue en 1222) puis autour de la Sorbonne (1253), avec Saint Albert le Grand(vers 1200-1280), Saint Thomas d'Aquin (1224-1274) et tant d'autres (sans oublier le précurseur Abélard, 1079-1142) jetait les bases de la pensée laïque occidentale, et si Molière et ses contemporains ont pu à leur époque, dans *Le Malade imaginaire* par exemple, ridiculiser la scolastique, alors certes dépassée, on n'aurait garde de sous-estimer la révolution intellectuelle capitale qu'elle a constituée.

Une analyse possible de l'essor de la civilisation occidentale par rapport à ses concurrentes orientales chrétienne autant que musulmane compte au nombre des facteurs favorables la persistance de cette diversité intellectuelle, alors que tant la chrétienté orientale que l'Islam, à peu près simultanément vers le XIIe siècle, ont figé la pensée religieuse et interdit toute interprétation d'une doctrine désormais supposée autosuffisante et intangible. Tout cela est bien sûr hypothétique et demanderait des nuances ; la tolérance à l'égard des penseurs indépendants en Occident n'est pas allée sans de nombreux bûchers où brûlèrent des hérétiques et leurs écrits, cependant que des penseurs arabes et grecs ont trouvé mille moyens de s'exprimer en bravant ou en contournant des interdits dont la vigueur était d'ailleurs variable selon l'ambiance de l'époque ou les traits de personnalité des autorités du moment. Il reste qu'après le XIIe siècle l'interrogation philosophique se tarit dans les mondes musulman et orthodoxe, la philosophie s'y fige en « une simple mise en ordre logique du monde social donné, acquise une fois pour toutes » (Cornelius Castoriadis [26], p. 119). Il reste aussi que l'on a pu dire que la pensée occidentale avait pour caractéristique originale d'être essentiellement mue par le doute, qui semble s'être avéré meilleur pilote que la certitude.

Anthropologie (3) :

Les structures anthropologiques ont une pérennité supérieure à celle des structures sociales, les sources archéologiques attestent la permanence de structures familiales depuis le néolithique, mais rien n'interdit de penser à des origines encore plus anciennes. Or, on connaît la thèse de Friedrich Engels [39] qui prête à toutes les sociétés préhistoriques voire antiques ou médiévales une organisation en familles élargies qui regrouperaient autour d'un ancêtre vivant ou disparu plusieurs générations et plusieurs ensembles matrimoniaux (la polygamie éventuelle interdit de parler de couples) et leurs descendances. Toujours selon Engels, la famille nucléaire, c'est-à-dire réduite à un couple de parents et à ses enfants, serait apparue avec les temps modernes, le développement des villes et les débuts du capitalisme. Curieusement, alors que le moins que l'on puisse dire de cette thèse d'Engels est qu'elle n'est pas le mur le plus solide de l'édifice marxiste, jusqu'à une époque récente les spécialistes en sciences humaines de toutes tendances l'avaient pour leur majorité adoptée au moins tacitement, et elle survit fort bien dans les représentations populaires. L'anthropologie moderne l'a réfutée, en établissant sur des bases archéologiques et documentaires l'ancienneté de l'organisation familiale nucléaire des populations, par exemple, de la France septentrionale et de l'Angleterre. Les travaux d'Emmanuel Todd et Hervé Le

Bras [22] donnent une présentation générale de ces résultats et en tirent des conséquences intellectuellement stimulantes au-delà de l'anthropologie.

Ce n'est pas seulement la philosophie des Grecs que l'Occident a redécouverte par le truchement des Arabes, mais aussi leur science. L'imprimerie à partir du XVe siècle a amplifié la portée de ces innovations. Les bouleversements auxquels l'imprimerie a fortement contribué n'ont pas besoin d'être rappelés, au premier rang desquels la Réforme et ses conséquences philosophiques, sociales et politiques.

Il faut dire que le christianisme, en développant une conception originale (quoique non dépourvue de racines juives et grecques) de la personne, allait donner son essor à un être presque inédit, l'individu. Dans la plupart des sociétés traditionnelles, l'individu n'est rien séparé du groupe familial, clanique, tribal ou autre auquel il appartient. Le christianisme, religion de femmes et d'esclaves, née dans la clandestinité, a voulu soustraire les prosélytes à l'autorité du pouvoir politique, et ce faisant a inventé la notion de statut personnel. Cette situation historique singulière d'une religion séparée (du moins à son origine) du pouvoir politique a engendré des distinctions et des catégories inédites : les distinctions entre pouvoir temporel et pouvoir spirituel, entre clercs et laïcs, et la notion même de laïcité n'ont de sens que dans un contexte chrétien. Un épisode oublié du combat séculaire de l'Église pour la liberté individuelle concerne le mariage, qui en France sous l'ancien régime était soumis à l'autorisation des pères des deux époux, quel que soit leur âge : l'Église estimait que le prêtre était le seul à même de juger de l'opportunité de prononcer ce sacrement, et elle luttait pour la liberté de choix des futurs conjoints, liberté bien sûr guidée par l'obéissance du fidèle à son curé. La Révolution tranchera le conflit.

Cela soulève bien sûr la question de la liberté. Aujourd'hui cette dernière est souvent invoquée à tort et à travers, comme « la liberté de faire ce que je veux ». La liberté qui nous importe, au nom de laquelle des hommes ont sacrifié leur vie, ce n'est pas cela, cela n'a rien à voir avec ce fantasme infantile de toute-puissance. Il s'agit des libertés fondamentales : d'abord ne pas être esclave, bien sûr, puis liberté de conscience et d'expression, liberté d'aller et venir.

Le monde occidental ou occidentalisé vit aujourd'hui une époque qui proclame la liberté d'expression et exalte l'originalité et la créativité, mais il n'est guère besoin de creuser profond ou de visiter des contrées exotiques pour retrouver l'exigence unanimiste et conformiste. Nous ne saurions oublier que beaucoup de sociétés ressentent aujourd'hui la diversité d'expression comme nous la ressentions hier, comme une menace mortelle, et que cette perception n'est pas forcément déraisonnable même si elle n'a plus guère d'actualité pour nous. Que l'on songe simplement au déferlement de discours chauvins en 1914 : n'y a-t-il pas là motif d'interrogation sur notre propre société ?

Et d'ailleurs, cette liberté que nous plaçons aux frontons de nos bâtiments publics, en est-il fait si largement usage ? Et serait-ce souhaitable ? Une société où chacun pratiquerait dans la vie quotidienne le doute socratique serait-elle vivable ?

C.9 La brève trajectoire du Surmoi

Dans les sociétés traditionnelles où l'individu n'existe que par son appartenance à un groupe, les normes de pensée et d'action lui sont imposées de l'extérieur, le libre arbitre n'existe que pour des individus exceptionnels, généralement d'ailleurs considérés comme des fous ou des criminels et à ce titre réprimés par le corps social. Les étapes de la vie sont également fixées par le groupe : c'est la cérémonie d'initiation qui fixe l'instant où chacun devient adulte, et de même le mariage est fixé par des règles sociales assez indifférentes aux penchants individuels.

Le psychiatre Daniel Marcelli [79] ne manque pas de faire remarquer qu'une telle organisation sociale, dont l'emprise, si elle s'exerçait sur nous, nous semblerait à tout le moins désagréable, ne laisse pas de procurer à ses membres une place dans le groupe et une protection, sources d'une plénitude et d'une sérénité que beaucoup de nos contemporains pourraient leur envier.

Nous l'avons évoqué ci-dessus, l'individu apparaît pour devenir ce que la philosophie occidentale a nommé le sujet. Michel Foucault a entrepris d'en écrire l'histoire, tâche dont il n'oublie pas de signaler qu'elle apparaît suspecte tant aux historiens qu'aux philosophes. Le sujet se manifeste d'abord à l'état isolé, dès l'Antiquité grecque et romaine. Le christianisme poursuit son effort pour soustraire le fidèle à son environnement temporel. La Renaissance italienne étend la redécouverte de l'Antiquité aux arts, fonde l'humanisme et, avec Masaccio notamment au début du XVe siècle, place l'homme au centre de l'interrogation artistique, place précédemment occupée par Dieu. C'est René Descartes, par son affirmation impérieuse *Cogito ergo sum*, qui conférera au sujet la domination de la scène intellectuelle, avant qu'il n'investisse la société tout entière.

Ce développement de l'individu en sujet aux XVIIe et XVIIIe siècles va de pair avec la sécularisation de la société, c'est-à-dire le retrait de la religion de la sphère publique vers la sphère privée. Dans un pays comme la France, la croyance et la pratique religieuse sont des affaires strictement privées, et nous voyons bien que toute irruption de la religion sur la scène publique déclenche des réactions très négatives. Nous ne devons pas oublier que cet état de fait est d'une part récent, d'autre part très peu universel — en effet il n'en va pas de même dans de nombreuses régions du monde.

L'essor du sujet et son apogée sous les Lumières ont bien sûr joué un rôle capital, voire déterminant, dans les événements politiques de cette époque, nommément les révolutions américaine et française et leurs suites. Il s'ensuivit l'instauration dans le monde occidental de régimes démocratiques ou revendiqués comme tels. Rappelons que la démocratie est une forme politique (cf. note 2 p. 154), c'est-à-dire un moyen et non une fin ; la fin visée par ce moyen est, pour rester assez général, la liberté humaine, qui, elle, est une valeur universelle, alors que la démocratie est une forme dont l'usage est (pour l'instant du moins) limité à un certain univers culturel (le monde occidental tel que nous l'avons évoqué ci-dessus, dont les habitants ont vécu l'ascension du sujet, la sécularisation et l'individuation), et aux institutions politiques. Si l'idéal universel de liberté humaine a sa place au sein des entreprises ou des familles, par exemple, il n'est pas pour autant pertinent de vouloir y introduire la démocratie, qui est d'un autre ordre (sauf évidemment pour les secteurs du fonctionnement de ces institutions qui relèvent du politique, comme par exemple les élections des représentants du personnel).

Cette souveraineté du sujet sur sa propre pensée et sur sa propre psyché sera de courte durée. La psychanalyse et la sociologie vont s'attacher à montrer que les instances de contrôle de l'individuel par le collectif, dont les Lumières croyaient avoir définitivement triomphé, n'avaient opéré qu'une courte migration, et ces deux disciplines vont s'ingénier à ruiner les illusions du libre arbitre, l'une en révélant l'empire de l'inconscient sur nos actes que nous croyions les plus libres, l'autre en mettant en évidence la banalité statistique de nos plus superbes originalités, parfois jusqu'à suggérer leur orchestration par quelque détermination sociale.

« La norme collective passe de l'extérieur à l'intérieur du sujet et cette autorité intériorisée de la coutume et des dieux porte un nom : le Surmoi » écrit Daniel Marcelli [79] (p. 216).

Cette instance intérieure d'où émanent conscience morale, interdits et idéaux crée en chaque individu une situation conflictuelle. La canalisation et la répression des pulsions, l'interdiction de désirs coupables, la génération de désirs légitimes, effectuées dans les sociétés traditionnelles par des instances externes, sont désormais des productions endogènes. Chacun est alors la proie de désirs contradictoires que le moi résoudra par des mécanismes de défense : le refoulement, « qui consiste à renvoyer dans la part inconsciente du fonction-

nement psychique des pulsions, des désirs, des représentations, des affects ou des émotions dont la nature ou le but font l'objet d'une réprobation du moi ou du Surmoi » ou le clivage, « aboutissant en quelque sorte à ce que la main droite ignore ce que fait la main gauche » [79] (pp. 218-219). Le refoulement qui n'a pas pu être déplacé ou sublimé pourra engendrer souffrances et névroses, le clivage qui met en péril la cohérence de l'individu est au principe des psychoses et des addictions.

Marcel Gauchet [46] distingue une personnalité « traditionnelle » d'une personnalité « moderne ». Il écrit : « Le monde de la personnalité traditionnelle est un monde sans inconscient en tant qu'il s'agit d'un monde où le symbolique règne de manière explicitement organisatrice... L'être individuel d'avant l'individualisme est littéralement constitué par la norme collective qu'il porte en lui ». La personnalité moderne se présente au contraire comme « l'âge d'or de la conscience et de la responsabilité, âge d'or dont on voit comment il comporte comme contrepartie logique la mise en évidence d'un inconscient où se réfugie la part symbolique qui n'a plus sa place dans le fonctionnement collectif où les règles de droit remplacent l'autorité de la coutume et des dieux » (p. 251).

Le Surmoi a donc une histoire (il apparaît au milieu du XIXe siècle) et une géographie (sa première capitale fut sans conteste Vienne, d'où il rayonna sur le monde occidental, sans oublier l'Argentine, avec une prédilection pour les régions urbanisées). Les observateurs ne l'ont pas encore repéré sous d'autres cieux. Ainsi Marie-Cécile et Edmond Ortigues [88] écrivent à propos du Sénégal des années 1970 : « L'agressivité s'exprime principalement sous la forme de réactions persécutives. La culpabilité est peu intériorisée ou constituée comme telle... Tout se passe comme si l'individu ne pouvait pas supporter de se percevoir divisé intérieurement, mobilisé par des désirs contradictoires. Le "mauvais" est toujours situé à l'extérieur de moi, il est du domaine de la fatalité, du sort, de la volonté de Dieu. » (pp. 110-113).

Mais il ne faudrait pas en déduire qu'une société sans Surmoi est une société sans morale, bien au contraire : simplement, les procédés de fabrication de morale sont différents et localisés en d'autres instances.

Le Surmoi est aujourd'hui en péril. Daniel Marcelli observe parmi ses patients (il est pédopsychiatre) des enfants dont les parents se sont ingéniés à éviter toute situation où ils auraient pu avoir avoir à leur dire « Non ! ». Ce principe éducatif achoppe bien sûr dès l'entrée à l'école maternelle, voire dès la naissance d'un petit frère ou d'une petite soeur. Ces pratiques éducatives contemporaines ne permettent pas à leurs victimes d'élaborer un Surmoi bien constitué, et comme par ailleurs les anciennes instances régulatrices ont disparu ou sont affaiblies, l'individu se heurte avec souffrance et incompréhension à une société qui n'a pas subitement acquis une tolérance infinie à l'égard de ses fantasmes de toute-puissance et de désirs dont il attend la satisfaction sans délai. Le pronostic du docteur Marcelli n'est pas optimiste : psychoses et addictions (substances psychotropes, fièvre acheteuse, jeux vidéo...) attendent ces individus déboussolés. L'idéologie de l'épanouissement individuel inconditionnel et à tout prix ne mène pas forcément au bonheur radieux.

C.10 Fonction publique

Il semble que pour certains agents des services publics la difficulté à appréhender la nature du travail qui leur est demandé soit encore accrue par une incertitude quant à la source légitime de la demande de travail. En apparence, ils ont bien un employeur, qui en dernière instance est le gouvernement (ou une collectivité locale), dûment mandaté par les représentants élus du peuple souverain. Ce que les fonctionnaires doivent faire est décidé non par eux-mêmes, mais par lesdits représentants élus du peuple souverain, qui en discutent à l'Assemblée Nationale et votent, notamment, la Loi de Finances qui instaure des dispositions selon lesquelles les contribuables vont payer des impôts destinés, comme le

mot l'indique, à contribuer à des tâches dont le législateur (c'est-à-dire les représentants élus du peuple souverain) a reconnu l'utilité et que des fonctionnaires vont devoir accomplir. Pour que cela puisse se faire avec un peu de méthode, certains fonctionnaires se voient confier des responsabilités et éventuellement une autorité destinée à s'exercer sur d'autres fonctionnaires afin d'organiser et de réaliser le travail. Bref, on ne voit aucune raison pour que le fonctionnement des services publics, sous cet angle de l'organisation du travail, diffère fondamentalement de celui d'une entreprise, et pour que les fonctionnaires soient exemptés de l'obligation d'accomplir, en échange de leur traitement, le travail que leur a assigné leur employeur.

Je n'ignore pas que certaines catégories de fonctionnaires ne sont pas formellement soumises à une autorité hiérarchique issue en dernier recours du pouvoir législatif. Il s'agit des magistrats, forts de l'indépendance de la justice, des enseignants du cycle supérieur qui bénéficient des franchises et de l'autonomie des Universités, et des chercheurs relevant des statuts instaurés, en France, par la loi d'orientation de la recherche en 1982. Ces deux dernières catégories relèvent du processus d'« évaluation par les pairs », qui les soustrait (en principe) à l'autorité administrative tout en leur conférant un statut de fonctionnaire et la garantie qui en découle de l'emploi à vie ; les membres de ces corporations sont élus par leurs futurs collègues réunis en commissions qui élaborent elles-mêmes leurs propres critères de recrutement et de promotion, cependant que le seul moyen de pression que possède sur eux l'autorité administrative (mais il n'est pas mince) réside dans l'attribution sélective de crédits, postes et autres moyens matériels. Dans les disciplines où l'agitation compétitive et la course aux crédits sont fébriles, comme la biologie moléculaire par exemple, il n'est pas sûr que les contraintes subies par les chercheurs ne soient pas plus lourdes que dans une administration classique ; il reste qu'elles apparaissent moralement moins désagréables aux intéressés.

Les lignes qui précèdent conduisent l'auteur à prendre le contre-pied d'une idée pas totalement absente de l'esprit de certains fonctionnaires : l'argent gagné par les employés des entreprises privées serait sale et répugnant, parce que fruit d'une activité lucrative forcément plus ou moins apparentée au vol ou à l'escroquerie, alors que le traitement du fonctionnaire serait propre, parce que tombé en quelque sorte du ciel. Ne parlons même pas des gains des commerçants, artisans et autres entrepreneurs : les adeptes de cette croyance n'ont pas de nausée assez dégoûtée pour y penser ! Une petite enquête auprès des commerçants du secteur alimentaire de son quartier, relative notamment aux horaires de travail, a convaincu l'auteur que leur revenu horaire était bien inférieur à celui de salariés au SMIC ; la vraie richesse éventuelle d'un commerçant, c'est son fonds de commerce, dont la revente pourra lui tenir lieu de retraite, à condition que la valeur ne s'en soit pas trop dégradée, et dans la mesure où les retraites assurées par les caisses des travailleurs indépendants sont très faibles. Dans le cas des petits commerces d'alimentation, par exemple, les espoirs de revente sont aujourd'hui très décourageants, pour ne rien dire des bureaux de tabac…

Une conséquence de la mauvaise opinion nourrie par les intellectuels du service public à l'égard des revenus de l'industrie et du commerce (notons au passage que l'agriculture ne partage pas cette désaffection) se fait sentir à l'encontre des chercheurs qui partent dans le privé. Ainsi, j'ai pu suivre l'itinéraire d'un ingénieur qui avait développé au sein d'un établissement de recherche publique un logiciel innovant dans le domaine bioinformatique. Durant des années il avait postulé, en vain, au poste de chercheur qu'il méritait sans doute. Face aux fins de non-recevoir de son organisme, il se décida à créer une entreprise pour que lui et ses assistants puissent vivre des résultats de leur travail. J'ai recueilli les opinions émises à son sujet par des chercheurs qui, eux, avaient obtenu un poste, tout en n'ayant jamais fourni de travail comparable : mépris, propos insultants, rejet, refus d'utiliser ce logiciel vénal…

Dans tous les cas que j'ai observés de fonctionnaires de la recherche qui ont quitté (parfois sans avoir réellement le choix) le dispositif public pour un employeur privé, ils

se sont vu accuser d'adorer le veau d'or, même si c'était manifestement faux en termes d'enrichissement. Paradoxalement d'ailleurs, l'opprobre qui frappe le simple ingénieur ou chargé de recherche qui veut simplement gagner correctement sa vie ne frappe pas avec la même intensité le mandarin de haut grade qui cumule les postes honorifiques et lucratifs dans les conseils scientifiques de grands groupes industriels.

Cet aveuglement de certains fonctionnaires sur leur propre position sociale, sur les conditions d'exercice de leur activité et sur celles qui assurent leur survie économique n'est pas sans évoquer le texte de Franz Kafka [60] « Recherches d'un chien » et l'analyse fameuse qu'en a faite Gilles Deleuze [35]. Ce texte écrit à la première personne a pour narrateur un jeune chien urbain de la première moitié du XXe siècle, c'est en quelque sorte un petit *Bildungsroman* dont le héros, Wilhelm Meister canin, décrit la complexité du monde des chiens, sa variété sociale et la façon dont il a réussi à s'y orienter en s'inspirant de l'exemple de vieux chiens particulièrement sages et avisés. La vie des chiens domestiques est bien sûr fortement conditionnée par le contrôle qu'exercent sur eux leurs maîtres humains, mais le narrateur ignore systématiquement cette détermination, et l'aveuglement culmine lorsqu'il en vient à parler des chiens de manchon, décrits comme s'ils planaient dans l'air sans intervention extérieure. Deleuze estime que Kafka a voulu ici évoquer certains milieux juifs qu'il pouvait fréquenter et dont la vision très introvertie ignorait excessivement le monde non juif qui les entourait. De même certains fonctionnaires enfermés dans leur tour d'ivoire (surtout si elle est universitaire ou académique au sens large) ignorent trop le monde extérieur dont les contributions matérielles assurent leur subsistance, ignorance qui peut conduire à des déconvenues.

C.11 Le travail est (aussi) une base de la société

Nous avons signalé comment la bourgeoisie en ascension des temps modernes avait fait du travail une valeur économique, une marchandise, et nous avons essayé de rappeler brièvement les conséquences de cette conception radicalement novatrice sur l'organisation de notre société et la position en son sein des individus qui la constituent. Nous ne saurions terminer ce chapitre sans corriger la vision quelque peu unilatérale que le lecteur pourrait retirer des paragraphes ci-dessus. Les chapitres de ce livre abordent certaines formes très modernes, très nouvelles du travail, nous y montrons que les anciens modèles de mesure de la quantité de travail, hérités de l'industrie du XIXe siècle, s'y appliquent mal. Pour replacer le travail contemporain dans son contexte et ainsi essayer de le comprendre, il faut évoquer ses aspects non économiques.

Le travail n'est pas seulement une marchandise. Il est aussi, et nous l'avons déjà noté, une valeur morale. Aujourd'hui, dans un pays comme la France, le travail est partie intégrante de l'identité des individus, il est un des moyens principaux de leur socialisation, ceux qui en sont exclus pour une raison ou une autre sont soumis à la déréliction, à la souffrance et à la désocialisation.

S'il est dans la logique des entreprises de considérer le travail comme une marchandise, d'autres points de vue peuvent exister dans la société et même dans l'entreprise. C'est le propre d'une organisation sociale complexe que chaque phénomène puisse être appréhendé sous des angles divers par ses acteurs, d'ailleurs eux-mêmes porteurs de points de vue différents selon les rôles multiples qu'ils peuvent jouer (travailleur, parent d'élève, électeur, contribuable...). Si la logique marchande peut sembler à court terme la plus profitable aux entreprises, si rien ne vient la contrebalancer, elle révèle à plus long terme des inconvénients qui ont leur coût. La logique de l'entreprise n'est d'ailleurs pas purement marchande. L'entreprise baigne dans l'espace marchand, qui lui est extérieur ; mais dans son intérieur les relations ne sont pas marchandes, mais organisées. On pourra consulter sur le site de Michel Volle une réflexion

originale sur la nature de l'entreprise, sous le titre *Qu'est-ce qu'une entreprise ?* (13 octobre 2003).

On pourra à ce sujet se reporter au livre de Stéphane Beaud et Michel Pialoux [8], qui décrit, dans un secteur de la société et de l'économie assez éloigné de celui qui sera envisagé dans la suite de cet ouvrage (l'usine Peugeot de Sochaux et les usines sous-traitantes dans la région de Montbéliard), les effets dévastateurs d'une modernisation industrielle aveugle et la destruction qu'elle impose aux facteurs de cohésion sociale qui permettent tout simplement aux gens de vivre. Ce livre montre notamment que les conditions très dures du travail à la chaîne qui avaient été vigoureusement critiquées dans les années 1970, et que le discours managérial et le recul des luttes ouvrières dans l'industrie avaient fait oublier, existent de plus belle, simplement les travailleurs qui y sont soumis sont placés dans une position d'individualisation du travail qui interdit toute réaction collective (nous y reviendrons au chapitre suivant). Cette situation est largement le fruit du contournement des législations sociales par les employeurs (notamment par le recours systématique à l'emploi intérimaire et à la sous-traitance), qui aboutit à une dérégulation de fait. Elle contribue à une désagrégation sociale qui a certes aussi d'autres causes mais dont le coût (pas seulement financier) est élevé, pour ne pas dire intolérable.

L'irruption sur la scène sociale de la question longtemps occultée du harcèlement moral au travail montre que les entreprises du secteur tertiaire, au personnel composé en forte proportion d'ingénieurs et de cadres bien rémunérés, ne sont pas à l'abri des effets délétères de certaines orientations managériales.

Index

Bibliographie

[1] Harold Abelson, Gerald Jay Sussman et Julie Sussman. *Structure and Interpretation of Computer Programs*. Cambridge, Massachusetts : MIT Press, 1996. URL : https://web.mit.edu/6.001/6.037/sicp.pdf.

[2] Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge : Cambridge University Press, 1996.

[3] AFAQ. "Association Française de l'Assurance Qualité". In : *Site de l'AFAQ* (2004). URL : http://www.afaq.fr.

[4] AFNOR. *Certification ISO 9000*. Paris : AFNOR, 2001.

[5] AFNOR. *Qualité et systèmes de management ISO 9000*. Paris : AFNOR, 2001.

[6] Hannah Arendt. "La crise de la culture, huit exercices de pensée politique". In : Paris : Gallimard, 1972 – 1989. Chap. *Qu'est-ce que l'autorité ?*

[7] Mohamed Arkoun. *La pensée arabe*. Paris : Presses Universitaires de France – Que sais-je ?, 1975.

[8] Stéphane Beaud et Michel Pialoux. *Violences urbaines, violence sociale*. Paris : Fayard, 2003.

[9] Jean-Louis Bénard et al. *L'eXtreme Programming*. Paris : Eyrolles, 2002.

[10] Didier Bert, Henri Habrias et Véronique Viguié Donzeau-Gouge. "Méthode B (numéro spécial)". In : *Technique et science informatique* 22.1 (2003).

[11] Laurent Bloch. *Révolution cyberindustrielle en France*. Paris : Economica, 2015. URL : https://laurentbloch.net/MySpip3/Revolution-cyberindustrielle-en-307.

[12] Laurent Bloch. *Initiation à la programmation et aux algorithmes avec* Python. Paris : Technip, 2020.

[13] Laurent Bloch. *Initiation à la programmation et aux algorithmes avec* Scheme. Paris : Technip, 2020.

[14] Laurent Bloch. *Splendeurs et servitudes des systèmes d'exploitation – Histoire, fonctionnement, enjeux*. Paris : Laurent Bloch, 2020. URL : https://laurentbloch.net/MySpip3/Systeme-et-reseau-histoire-et-technique.

[15] Laurent Bloch et al. *Rapport d'activité du Service d'Informatique Scientifique – Institut Pasteur*. Rapp. tech. 17 novembre 1998. URL : https://laurentbloch.net/Data/RapportPasteur/Html/index.html.

[16] Laurent Bloch et al. *Sécurité informatique – Pour les DSI, RSSI et administrateurs*. Paris : Eyrolles, 2016.

[17] Grady Booch, James Rumbaugh et Ivar Jacobson. *UML – Guide de l'utilisateur*. Paris : Eyrolles, 2000.

[18] Mireille Boris. "La formation à l'heure des "nanopédagogies"". In : *Stic-Hebdo* (26 janvier 2004). URL : http://www.asti.asso.fr/pages/Hebdo/sh04/sh04.htm/.

[19] Jean-Pierre Boutinet. *Anthropologie du projet*. Paris : Presses Universitaires de France, 1990 – 2003.

[20] Bouygues Télécom. "Relation clients". In : (6 mai 2004). URL : http://www.recrute.bouyguestelecom.fr/relationclient.html.

[21] Isabelle Boydens. *Informatique, normes et temps*. Bruxelles : Bruylant, 1999.

[22] Hervé Le Bras et Emmanuel Todd. *L'invention de la France*. Gallimard, 2012.

[23] Philippe Breton. *La tribu informatique – Enquête sur une passion moderne*. Paris : Métailié, 1991.

[24] Frederick P. Brooks, Jr. *The Mythical Man-Month (*traduction *: Le mythe de l'homme-mois)*. Reading, Massachusetts (Paris) : Addison-Wesley (Vuibert pour la traduction), 1975-1995.

[25] Martin Campbell-Kelly. *Une histoire de l'industrie du logiciel*. Cambridge, Massachusetts (Paris) : MIT Press (Vuibert pour la traduction française de Pierre-Éric Mounier-Kuhn), 2003.

[26] Cornelius Castoriadis. *La montée de l'insignifiance*. Paris : Le Seuil, 1996.

[27] Cornelius Castoriadis. *Ce qui fait la Grèce, 1 : D'Homère à Héraclite*. Le Seuil, 2004.

[28] Yves Chaimbault. "Marchés publics : le nouveau nouveau code". In : *Revue des Secrétaires Généraux* (septembre 2004). URL : http://www.cpu.fr/Publications/Publication.asp?Id=336.

[29] Henri Chelli. *Urbaniser l'entreprise et son système d'information*. Avec une préface de Michel Volle. Paris : Vuibert, 2003.

[30] Société ClearSy. "Atelier B". In : (juin 2004). URL : http://www.atelierb.societe.com/.

[31] Club des Maîtres d'Ouvrage. "Site du Club des Maîtres d'Ouvrage des Systèmes d'Information". In : (2004). URL : http://www.clubmoa.asso.fr/.

[32] Cour des Comptes. "Le système Socrate et la nouvelle tarification de la SNCF". In : (1996). URL : http://www.ccomptes.fr/Cour-des-comptes/publications/rapports/rp1996/cdc63_5.htm.

[33] Cour des Comptes. "Site de la Cour des Comptes". In : (2004). URL : http://www.ccomptes.fr/.

[34] Marie Coris. "Impact des logiciels libres sur l'industrie du logiciel : vers un nouveau modèle productif?" In : *Actes du congrès JRES*. Sous la dir. de Roland Dirlewanger. 2001. URL : http://www.jres.org/Archives/JRES2001/jres2001CD.pdf.

[35] Gilles Deleuze et Félix Guattari. *Kafka – Pour une littérature mineure*. Paris : Les Éditions de Minuit, 1975.

[36] Linda G. DeMichiel. *Enterprise JavaBeans Specification, version 2.1*. Santa Clara, Calif. : Sun Microsystems Inc., 12 novembre 2003.

[37] Marc Destors, Jean Le Bissonais et Marc Fersten. *Le Chef de projet paresseux... mais gagnant*. Les Ulis, France : Microsoft Press, 2000.

[38] Edsger Wybe Dijkstra. "The structure of the THE multiprogramming system". In : *Communications of the ACM (CACM)* 11.5 (mai 1968). URL : http://www.acm.org/classics/mar96/.

[39] Friedrich Engels. *L'origine de la famille, de la propriété privée et de l'État*. Paris : Costes (pour la traduction française), [1884] 1936.

[40] John Foderaro. “Lisp is a Chameleon”. In : *Site de Paul Graham* (1991). URL : http://www.paulgraham.com/chameleon.html.

[41] George Forsythe. “Computer science and education”. In : *Information processing* vol. 68 (1969).

[42] Gottlob Frege. *Écrits logiques et philosophiques*. Paris : Le Seuil, 1971 [1879–1925].

[43] Richard Gabriel. “The Rise of *Worse is Better*”. In : *Lisp : Good News, Bad News, How to Win Big* (3 août 1993). URL : http://www.ai.mit.edu/docs/articles/good-news/subsection3.2.1.html.

[44] Erich Gamma et al. *Design Patterns – Catalogue de modèles de conception réutilisables*. Reading, Massachusetts (Paris) : Addison-Wesley (Vuibert pour la traduction française), 1995.

[45] Bill Gates. *Business @ the Speed of Thought*. Warner Bros, 1999.

[46] Marcel Gauchet. *La démocratie contre elle-même*. Paris : Gallimard, 2002.

[47] Sumantra Ghoshal et Christopher Bartlett. “L’Entreprise individualisée”. In : *Institut du Management*. Sous la dir. d’EDF-GDF. Paris : Maxima – Laurent du Mesnil, 1998.

[48] René Girard. *La violence et le sacré*. Paris : Grasset, 1972.

[49] René Girard. *Des choses cachées depuis la fondation du monde*. Paris : Grasset, 1978.

[50] Kurt Gödel et Jean-Yves Girard. *Le théorème de Gödel*. Paris : Le Seuil, 1989.

[51] Solveig Godeluck. *La géopolitique d’Internet*. Paris : La Découverte, 2002.

[52] Natalia Grabar et Thierry Hamon. “Les relations dans les terminologies structurées : de la théorie à la pratique”. In : *Revue d’intelligence artificielle* 18 (1/2004).

[53] Paul Graham. “Great Hackers”. In : *Site de Paul Graham* (juillet 2004). URL : http://www.paulgraham.com/gh.html.

[54] Éric Guichard. “L’Internet : mesures des appropriations d’une technique intellectuelle”. Thèse de doct. 2002. URL : http://barthes.ens.fr/atelier/theseEG.

[55] Jean-Claude Guillebaud. *La refondation du monde*. Paris : Le Seuil, 1999.

[56] Katie Hafner et Matthew Lyon. *Where Wizards Stay Up Late – The Origins of the Internet*. Londres : Pocket Books, 1996.

[57] Christian Huitema. *Et Dieu créa l’Internet*. Paris : Eyrolles, 1995.

[58] IETF. “*Requests for Comments*”. In : *RFC* (2017). URL : https://www.ietf.org/rfc/.

[59] Irène Inchauspé. “David-Weill accuse”. In : *Le Point* n° 1554 (28 juin 2002).

[60] Franz Kafka. “Recherches d’un chien”. In : *La muraille de Chine*. Paris : Gallimard, [1923 (?)] 1950.

[61] William Kent. *Data and Reality. Basic Assumption in Data Processing Reconsidered*. New York : Elsevier North Holland, 1981.

[62] Victor Klemperer. *LTI, la langue du III^e Reich*. Leipzig [Paris] : Reclam Verlag [Albin Michel pour la traduction française], 1975 [1996].

[63] Donald E. Knuth. *The Art of Computer Programming*. Reading, Massachusetts : Addison-Wesley, 1997 [1968].

[64] Jean Kott. *Que devons-nous craindre de l’intelligence dite artificielle ?* 2015. URL : https://www.laurentbloch.net/MySpip3/Que-devons-nous-craindre-de-l.

[65] Kenneth L. Kraemer et Jason Dedrick. “Dell Computer : Organization of a Global Production Network”. In : *Center for Research on Information Technology and Organizations* (13 février 2002). url : http://www.crito.uci.edu/GIT/publications/pdf/dell.pdf.

[66] Émile Landormy. *Moi, Émile Landormy, indépendant du 21ème siècle*. Paris : Téraèdre, 1997.

[67] Gérard Le Lann. “*The Failure of Ariane 5 flight 501*”. In : Site de l’Inria (1999). url : https://hal.inria.fr/inria-00073613/file/RR-3079.pdf.

[68] Hervé Le Bras. *Les limites de la Planète : mythes de la nature et de la population*. Paris : Flammarion, 1994.

[69] Nathalie Le Breton. “La nécessité d’adapter les nouvelles technologies à leurs utilisateurs : les enseignements du projet SOCRATE à la SNCF”. In : *La Cinquième* (Rencontre-débat du 10 juin 1998). url : http://www.cite-sciences.fr/francais/web_cite/informer/tec_met/universite/texte/telecharge/uoct9810.pdf.

[70] Jean-Pierre Le Goff. *La barbarie douce – La modernisation aveugle des entreprises et de l’école*. Paris : La Découverte, 1999.

[71] Jean-Pierre Le Goff. *La démocratie post-totalitaire*. Paris : La Découverte, 2002.

[72] Maurice Leenhardt. *Do Kamo – La personne et le mythe dans le monde mélanésien*. Paris : Gallimard, 1947.

[73] Légifrance. “Loi Sapin”. In : *Journal Officiel* (29 janvier 1993). url : http://www.legifrance.gouv.fr/texteconsolide/MEEAA.htm/.

[74] Josh Lerner et Jean Tirole. “The Simple Economics of Open Source”. In : *National Bureau of Economic Research* (2000). url : http://www.people.hbs.edu/jlerner/simple.pdf.

[75] Lawrence Lessig. *The future of ideas – The fate of the commons in a connected world*. New York : Random House, 2001.

[76] Émile Littré. *Dictionnaire de la langue française*. Paris : Hachette, 1865. url : https://www.littre.org/.

[77] A.S. Loebl. “Accuracy and Relevance and the Quality of Data”. In : *Data Quality Control. Theory and Pragmatics*. T. 112. Statistics : Textbooks and Monographs. New York : Marcel Dekker, 1990.

[78] Corinne Maier. *Bonjour paresse – De l’art et de la nécessité d’en faire le moins possible en entreprise*. Paris : Michalon, 2004.

[79] Daniel Marcelli. *L’enfant, chef de la famille*. Paris : Albin Michel, 2003.

[80] Jean-Pierre Meinadier. *Ingénierie et intégration des systèmes*. Paris : Hermès, 1998.

[81] Philippe Meirieu et Marc Guiraud. *L’école ou la guerre civile*. Paris : Plon, 1997.

[82] Bertrand Meyer. “The Ethics of Free Software”. In : *Software Development Magazine* (mars 2000). url : http://se.ethz.ch/~meyer/publications/softdev/ethics.pdf.

[83] Ministère de l'Économie et des Finances. “L’espace des Marchés Publics”. In : *Site du Ministère de l’Économie et des Finances* (2004). url : http://www.minefi.gouv.fr/minefi/publique/publique4/index.htm.

[84] Christian Morel. *Les décisions absurdes – Sociologie des erreurs radicales et persistantes*. Paris : Gallimard, 2002.

[85] Andrée Muller. “René Colletti (MC2I) : « C’est au bulldozer qu’il faut désormais aller dans les systèmes d’information »”. In : *01 Informatique* 1761 (19 mars 2004).

[86] Pierre-Alain Muller et Nathalie Gaertner. *Modélisation objet avec UML*. Paris : Eyrolles, 2003.

[87] John von Neumann. *The Computer and the Brain*. Traduction française : La Découverte, Paris 1992.
Ce texte est celui d'une conférence qui n'a pas été prononcée à cause de la mort brutale de l'auteur. Il réfute le réductionnisme qui fleurit souvent sous de tels titres, énumère les différences fondamentales entre l'activité du cerveau et le fonctionnement des machines numériques, ouvre de nouvelles problématiques sur des questions rebattues telle que l'existence des objets de la mathématique et de la logique. New Haven, Connecticut : Yale University Press, 1957.

[88] Marie-Cécile Ortigues et Edmond Ortigues. *Œdipe africain*. Paris : Union Générale d'Éditions, 1973.

[89] Roger Penrose. *L'esprit, l'ordinateur et les lois de la physique*. Paris : Interéditions (pour la traduction française de Françoise Balibar et Claudine Tiercelin), 1992.

[90] Pat Phelan et Ned Frey. "Avoiding Failure in Large IT Projects : New Risk and Project Management Imperatives". In : *Gartner Group* (2003). URL : http://www.gartnergroup.com/.

[91] Karl Popper. *La connaissance objective*. Paris : Aubier – Flammarion (pour la traduction française de Jean-Jacques Rosat), [1979] 1991.

[92] Jean-Marie Portal. "Les DSI dans le collimateur". In : *01 Informatique* 1774 (18 juin 2004). URL : http://www.01net.com/article/236912.html.

[93] Jacques Printz. *Coût et durée des projets informatiques*. Paris : Hermès, 2001.

[94] Jacques Printz. *Productivité des programmeurs*. Paris : Hermès, 2001.

[95] François Rastier. "Ontologie(s)". In : *Revue d'intelligence artificielle* 18 (1/2004). URL : http://www.revue-texto.net/Inedits/Rastier_Ontologies/Rastier_Ontologies.html.

[96] Dennis M. Ritchie. "The Evolution of the Unix Time-sharing System". In : *Lecture Notes in Computer Science* 79 (1980). Language Design and Programming Methodology.

[97] Claude Rochet. "Les enjeux stratégiques". In : (2003). URL : http://perso.wanadoo.fr/claude.rochet/fiches/info/enjeux.html.

[98] Claude Rochet et al. *L'intelligence iconomique – Les nouveaux modèles d'affaires de la troisième révolution industrielle*. De Boeck, 2015.

[99] *SAS Institute*. "Statistical Analysis System". In : (2004). URL : https://www.sas.com/.

[100] Moritz Schlick. *Forme et contenu*. Paris : Agone, [1936] 2003.

[101] Olivier Séhiaud. *DSI.CON – « L'informatique m'a tuer » (sic)*. Gossau – Suisse : Éditions 2020, 2004.

[102] Manuel Serrano. *Vers une programmation fonctionnelle* praticable. Une réflexion pratique non dépourvue toutefois d'aperçus théoriques sur la construction de logiciels. 2000. URL : https://www.laurentbloch.net/MySpip3/IMG/gz/hdr-serrano-ps.gz.

[103] Guy Sorman. *Les enfants de Rifaa*. Paris : Fayard, 2003.

[104] Richard Stallman. "GNU Coding Standards". In : *GNU Project* (20 février 2004). URL : http://www.gnu.org/prep/standards/.

[105] Standish Group. “The CHAOS Report”. In : *Standish Group* (1994). URL : http://www.standishgroup.com/sample_research/chaos_1994_1.php.

[106] Standish Group. “Unfinished Voyages – A Follow-Up to The CHAOS Report”. In : *Standish Group* (2003). URL : http://www.standishgroup.com/sample_research/unfinished_voyages_1.php.

[107] Joseph E. Stiglitz. *The Roaring Nineties (en français : Quand le capitalisme perd la tête)*. New-York (Paris) : W.W. Norton (pour la traduction : Arthème Fayard), 2003.

[108] Sun Microsystems. “Enterprise JavaBeans”. In : *Sun Developer Network Site* (25 août 2003). URL : http://java.sun.com/products/ejb/.

[109] Andrew S. Tanenbaum. *Architecture de l'ordinateur*. Paris : Dunod (pour la traduction française), 2001.

[110] Andrew S. Tanenbaum. *Réseaux*. Paris : Pearson Education (pour la traduction française), 2003.

[111] Andrew S. Tanenbaum. *Systèmes d'exploitation*. Paris : Pearson Education (pour la traduction française), 2003.

[112] Transparency International. “Indice de perception de la corruption 2003”. In : (2003). URL : http://www.transparency.org/.

[113] Alan Turing et Jean-Yves Girard. *La machine de Turing*. Une introduction abordable mais sans concessions, puis le texte historique. Paris : Le Seuil, 1995.

[114] Jean-Pierre Vernant. *Les Origines de la pensée grecque*. Presses universitaires de France, 1962.

[115] Michel Villette. *Sociologie du conseil en management*. Paris : La Découverte, 2003.

[116] Michel Volle. *Le métier de statisticien*. Ce livre, outre une introduction de première main aux questions et aux enjeux suscités par l'exercice de la statistique publique, introduit une réflexion originale sur la déontologie du spécialiste confronté à la mission d'informer le public non spécialisé. Paris : Economica, 1984. URL : http://www.volle.com/ouvrages/metier/tabmetier.htm.

[117] Michel Volle. *e-conomie*. Une analyse économique informée et pénétrante des nouvelles technologies par un des maîtres de l'économétrie et de la statistique. Paris : Economica, 2000. URL : http://www.volle.com/ouvrages/e-conomie/table.htm.

[118] Michel Volle. “Lettre ouverte à un dirigeant français”. In : (2001). URL : http://www.volle.com/opinion/lettre.htm.

[119] Michel Volle. “www.volle.com”. In : *Site de Michel Volle* (2004). URL : http://www.volle.com/.

[120] Michel Volle. *Analyse de données*. Paris : Economica, 1997 (4ème édition).

[121] Michel Volle. “Économie du système d'information”. In : (10 mai 2001). URL : http://www.volle.com/travaux/ecosi.htm.

[122] Michel Volle. “Histoire d'un *datawarehouse*”. In : (21 mars 2003). URL : http://www.volle.com/travaux/dwh.htm.

[123] Michel Volle. “Histoire d'un tableau de bord”. In : (20 novembre 2002). URL : http://www.volle.com/travaux/tdb.htm.

[124] Michel Volle. “Qu'est-ce qu'une entreprise ?” In : (13 octobre 2003). URL : http://www.volle.com/opinion/entreprise.htm.

[125] Michel Volle. “Recherche et pouvoir”. In : (26 mars 2004). URL : http://www.volle.com/opinion/recherche.htm.

[126] Michel Volle. “Sociologie des centres d’appel”. In : (20 mai 2001). URL : http://www.volle.com/opinion/centreappel.htm.

[127] Michel Volle. “Vocabulaire de l’informatique”. In : (22 avril 2004). URL : http://www.volle.com/ulb/021115/textes/vocabulaire.htm.

[128] Yair Wand et Richard Y. Wang. “Anchoring Data Quality Dimensions in Ontological Foundations”. In : *Communications of the ACM* 39.11 (novembre 1996), p. 85-95. URL : http://portal.acm.org/citation.cfm?id=240479&dl=ACM&coll=portal.

[129] Max Weber. *L’Éthique protestante et l’esprit du capitalisme*. Paris : Gallimard (pour la traduction française), [1905] 2003.

[130] Gerald M. Weinberg. *The Psychology of Computer Programming*. New York : Van Nostrand Reinhold, 1971.

[131] Encyclopédie Wikipedia. “Système formel”. In : *Wikipedia* (7 juillet 2005).

[132] Ludwig Wittgenstein. *Tractatus logico-philosophicus*. Paris : Gallimard (pour la traduction française), [1918].

[133] Philippe Zarifian. *À quoi sert le travail ?* Paris : La Dispute/Snédit, 2003.

[134] Alexandre Zinoviev. *Les hauteurs béantes*. Lausanne : L’Âge d’Homme (pour l’original et la traduction), 1976.