Organisation du travail à l'âge informatique

Laurent Bloch [1]

Mesurer, contrôler, payer le travail

La première révolution industrielle de la fin du XVIII^e siècle et surtout la seconde, de la fin du XIX^e (électricité et moteur à combustion interne), ont fait du travail une marchandise que des entreprises achètent à des travailleurs et dont il faut déterminer le prix.

Les économistes du XIX^e siècle ont beaucoup réfléchi à la valeur et au prix des marchandises. La valeur aurait pu être mesurée par l'utilité de la marchandise, mais l'utilité n'était pas la même pour tout le monde, et de toute façon on ne savait pas la mesurer. Alors, de Ricardo à Marx, tous sont tombés à peu près d'accord pour conclure que la valeur d'une marchandise était déterminée par la quantité de travail nécessaire à sa production, quantité elle-même mesurée par le temps de travail. Le temps de travail était facile à mesurer dans la grande entreprise de la seconde révolution industrielle (1875-1975) : horloges pointeuses à l'entrée des ateliers, contremaîtres sévères et chronométreurs tayloriens. Ce mode de production a atteint son apogée en France dans les années 1970, avec 40 % de la population active dans l'industrie, contre 20 % aujourd'hui [2].

Depuis quarante ans nous sommes entrés dans la troisième révolution industrielle, avec la généralisation de l'informatique permise par le microprocesseur à bas coût et la propagation du logiciel et de l'Internet. Sous ce régime toutes les tâches répétitives ont vocation à être automatisées, et le travail humain se concentrera dans les activités

1. lb@laurentbloch.org

2. http://michelvolle.blogspot.fr/2011/05/le-nouveau-monde.html Le blog de Michel Volle est une source particulièrement riche d'informations sur ces sujets.

mentales, intellectuelles ou psychologiques telles que la conception, l'ingénierie, les services, l'enseignement, les soins, la maintenance, etc. Les marchandises seront des assemblages complexes de biens et de services, élaborés par la coopération d'acteurs multiples dispersés dans le vaste monde, puisqu'aussi bien cette révolution cyberindustrielle [3] a permis la généralisation du conteneur, qui a divisé les coûts de transport par 50 [4].

Ainsi que le note Pascal Lamy dans l'article cité en référence, cette nouvelle ère économique remet en cause nos instruments de mesure de la production et du commerce : lorsqu'un téléphone portable franchit une douzaine de frontières dans l'un et l'autre sens avant d'arriver dans la vitrine du vendeur, que valent les statistiques du commerce extérieur ?

Il en va de même pour le travail : l'horloge pointeuse échoue à déterminer si le salarié assis devant son ordinateur crée de la valeur pour l'entreprise ou s'il bavarde sur Facebook, et les logiciels de surveillance ne serviront qu'à stimuler l'inventivité de ceux qui voudront les déjouer. Le temps de travail n'est plus une mesure pertinente de la valeur des biens et des services.

Notons que si la révolution cyberindustrielle nous prive de certains procédés de mesure de l'économie, elle nous en procure de nouveaux. Nous avons écrit ci-dessus que l'économie (classique, néo-classique, marxiste...) ne savait pas mesurer l'utilité d'un bien, et que de toute façon il aurait fallu pouvoir le faire pour chaque individu en particulier : ce problème est désormais résolu [5]. Amazon, Google, Uber savent parfaitement, en analysant mes navigations sur le Web et mes achats passés, déterminer les emplettes ou les trajets qui peuvent m'être suggérés. Et toutes les compagnies aériennes pratiquent le *yield management*, qui consiste à déterminer, à chaque date, pour chaque candidat au voyage, quel est le prix maximum qu'il est prêt à payer.

Nature du travail informatique

L'informatique transforme donc du tout au tout le travail, mais qu'en est-il du travail informatique proprement dit ?

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

3. Cf. Laurent Bloch, *Révolution cyberindutrielle en France*, 2015, Economica.

4. Cf. Pascal Lamy, « Mondialisation : renouveau conceptuel et renouvellement des politiques », *Commentaire*, mars 2012. Vol. 35, n° 137, pp. 118–126.

5. Cf. l'analyse de Michel Volle, Vincent Lorphelin et Christian Saint-Étienne parue dans *Le Monde* du 20 juin 2015, *Le triomphe de l'économie de l'utilité*, en ligne ici : `http://www.iconomie.org/wp-content/uploads/2015/06/le-monde-20-juin-2015.pdf`

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 [6]. 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.

C'est du milieu des années 1960, avec le lancement de la gamme IBM 360, que l'on peut dater la généralisation de l'informatique pour la gestion des entreprises, et du milieu des années 1970 la naissance d'une véritable industrie du logiciel, essentiellement tournée vers la gestion avant qu'elle n'enfourche le destrier de la micro-informatique des années 1980 pour peupler le monde de traitements de texte et de tableurs. Encore une douzaine d'années et nos écrans seront pleins des navigateurs de la seconde moitié des années 1990.

Dès lors il fallut bien se demander comment faire travailler les informaticiens, qui se comptaient par centaines de mille, puis par millions.

Le salut par Taylor ?

La grande industrie de la seconde révolution industrielle avait trouvé une doctrine avec Frederick Taylor et son organisation scientifique du travail [7]. Les managers de l'informatique, qui n'avaient pas conscience de vivre une révolution et même d'en être les acteurs, ont cherché à en transposer les idées dans leur activité.

Tout travail manuel, m'a appris un ami établi en usine après des études de philosophie et d'histoire et avant une reconversion à l'informatique, tout travail manuel, donc, repose sur une combinaison de contrainte physique et de conviction morale. C'est le fond du taylorisme, qui mise beaucoup sur la contrainte physique et peu sur la conviction morale. L'exemple extrême en est le travail militaire, où il faut donner

6. Martin Campbell-Kelly. *Une histoire de l'industrie du logiciel*. MIT Press (traduction française de Pierre-Éric Mounier-Kuhn), Cambridge, Massachusetts (Paris), 2003.

7. *The Principles of Scientific Management*, 1911.

sa vie : le texte extraordinaire du colonel Ardant du Picq explique bien comment ce sacrifice peut s'obtenir [8].

Contre toute vraisemblance et tout espoir, les managers ont misé sur la contrainte physique pour assurer la productivité des informaticiens : ateliers organisés comme les chaînes des usines automobiles, où chaque programmeur devait écrire ses 50 ou 100 lignes de code, puis les passer au suivant... Sous des formes moins caricaturales le rêve de l'usine informatique est encore bien vivant [9], c'est pourquoi nous allons examiner quelques façons curieuses d'envisager le travail informatique, afin de mettre au jour des erreurs qui s'y glissent et qui entraînent des échecs coûteux. Le lecteur qui vit dans le monde de la recherche pourrait croire ces exemples périmés : qu'il se détrompe, en beaucoup d'endroits ces méthodes aberrantes font encore florès.

Conception et réalisation

Assez logiquement, cette vision de la production informatique empruntait aussi à l'industrie automobile son modèle de division du travail : des ingénieurs conçoivent et dessinent des plans généraux, que des dessinateurs transforment en dessins détaillés, les ouvriers professionnels règlent les tours et les fraiseuses, les OS alimentent les machines. De même, des analystes concepteurs coucheraient sur le papier les spécifications du logiciel, avec un niveau de détail de plus en plus fin au fur et à mesure que l'on descendrait dans la hiérarchie, puis des programmeurs, ravalés au simple rang de codeurs, n'auraient plus qu'à écrire le programme, parfaitement déterminé par les spécifications.

Bien entendu, non seulement ce modèle a disparu de l'industrie automobile, mais il n'a jamais fonctionné dans l'industrie du logiciel. Plus exactement, il a été mis en pratique avec des résultats catastrophiques, en l'occurrence les millions de lignes de programmes Cobol dont personne ne sait plus depuis longtemps comment ils marchent ni même ce qu'ils sont censés faire exactement, mais qui règlent encore le fonctionnement de nos comptes bancaires et de nos contrats d'assurance.

Une vision managériale périmée : le cycle en V

Le cycle de développement « en V » hérité des métiers du bâtiment est encore préconisé par les méthodes traditionnelles. La figure 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 ».

8. *Études sur le combat*, 1880, `http://gallica.bnf.fr/ark:/12148/bpt6k864841`

9. De mon expérience de DSI j'ai tiré quelques procédés de diagnostic : dès que j'entends les locutions « outil informatique » ou « usine informatique » je sais que le cas de mon interlocuteur est sans espoir.

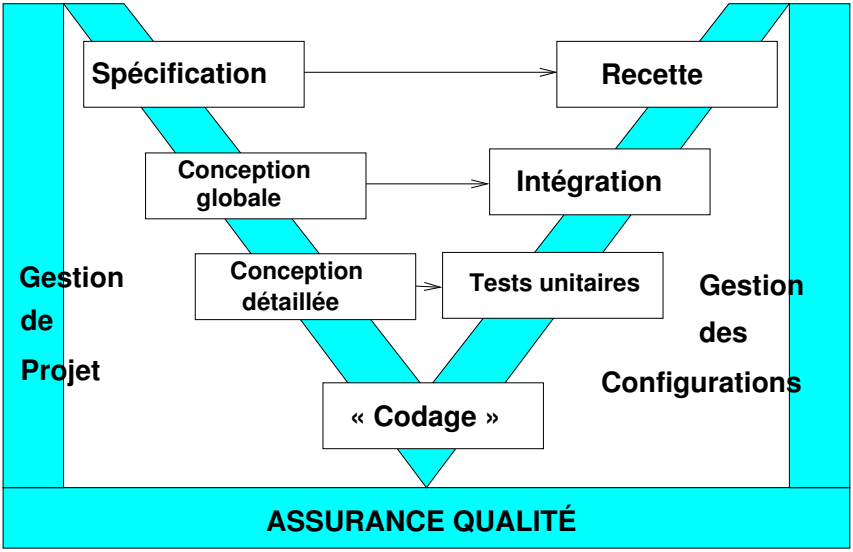

FIGURE 1. Cycle de développement « en V » du logiciel selon la vieille école.

Cette façon de faire est en voie d'abandon au profit des « méthodes agiles », plus à la mode et dont nous parlerons ci-dessous. Comment se propagent, puis reculent et disparaissent les méthodes de travail ? Les sociétés de services et les cabinets de consultants jouent un rôle capital dans ce processus, en effet ce sont eux qui implantent les méthodes nouvelles, d'abord chez leurs clients les plus riches et les plus dépensiers, puis de proche en proche jusqu'aux PME. Ce sont eux par exemple qui ont instauré la mode ruineuse des logiciels intégrés tels SAP et Oracle E-Business Suite, avant de leur substituer aujourd'hui les systèmes en nuage *(cloud computing)*. Il faut en effet savoir que si dans les années 1960 les entreprises qui s'informatisaient savaient qu'elles allaient devoir écrire des programmes et donc se doter de compétences adaptées, il y a quelques décennies que la mode est à l'externalisation, dont les managers espèrent souvent qu'elle leur permettra de renoncer à toute compétence informatique à l'intérieur de l'entreprise.

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 2015 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 soixante dernières années : l'écriture des systèmes d'exploitation.

En 1975 Brooks a écrit un livre intitulé *Le mythe de l'homme-mois* [10], 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 trente 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 ils ont abusé sans vergogne de leur pouvoir, créant par là les conditions d'un retour du bâton.

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

10. Frederick P. Brooks, Jr. *The Mythical Man-Month* (traduction de Frédéric Mora : *Le mythe de l'homme-mois*). Addison-Wesley (Vuibert pour la traduction), Reading, Massachusetts (Paris), 1975-1995.

pas, me semble une contribution de poids à la critique de la taylorisation informatique. Les universitaires ne sont pas très bien placés pour l'observer, mais cette approche taylorienne est encore très en vogue, je l'ai rencontrée il y a moins de cinq ans.

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 tâche 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. 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 tâches 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.

Je doute qu'une telle organisation ait été réellement mise en place pour beaucoup de 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 Web 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. Même des psychologues de la programmation comme Gerald Weinberg n'ont pas vraiment élucidé ce phénomène.

Ce qui fait travailler les programmeurs

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.

De façon plus générale, ce qui fait travailler les programmeurs, et avec eux de plus en plus de travailleurs de cette nouvelle économie que l'Institut de l'Iconomie [11] a nommée, justement, l'*iconomie*, c'est ce que Michel Volle appelle le *commerce de la considération* [12] :

> « La considération est cruciale aujourd'hui parce que le travail est devenu essentiellement mental : une entreprise qui n'écoute ni ses concepteurs, ni sa ligne de service, ne peut pas connaître les besoins des clients ni réussir ses innovations.
>
> Le cerveau d'un concepteur ne sera en effet productif que si celui-ci peut discuter ses idées avec les autres métiers de l'entreprise, avec les dirigeants. Un concepteur que l'on n'écoute pas a tôt fait de se renfermer dans sa coquille. Il y fera des choses qui l'intéressent, l'amusent ou répondent à une mode parmi les chercheurs, mais une bonne idée ne peut être féconde que si elle est adoptée et mise en pratique par l'entreprise. Le manque de considération envers les concepteurs stérilise la conception et inhibe l'innovation.
>
> [...]

11. Cf. `http://www.iconomie.org/`
12. Cf. `http://michelvolle.blogspot.com/2009/12/pour-un-de-la-consideration.html`

La considération est un *commerce* : il ne s'agit ni de moralisme, ni de sentimentalisme, mais d'un *échange* et comme tout échange celui-ci doit être *équilibré*. »

Méthodes agiles : *eXtreme Programming*

La méthode *eXtreme Programming* [13] (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 » et jouissent d'un certain succès après qu'aient été essuyés les échecs des méthodes plus rigides évoquées ci-dessus.

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 ;

13. Cf. *Gestion de projet eXtreme Programming*, Jean-Louis Bénard *et al.*, Eyrolles, Paris, 2004.

— 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 [14]. 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

14. 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.

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 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 notre expérience 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. Plaidons pour des logiciels moyens et la sobriété des projets.