Introduction

Une analogie circule dans les milieux informatiques, mais aussi dans le grand public, depuis les années 1970 : on dit que si l'industrie aéronautique avait fait les mêmes progrès que l'industrie informatique, les avions pèseraient aujourd'hui sept grammes, voleraient à mach 20, coûteraient le prix d'un vélomoteur et consommeraient trois décilitres de kérosène aux cent miles (les chiffres varient selon les époques).

L'analogie est bonne mais incomplète. Si l'aéronautique avait suivi les progrès de l'informatique, les avions supersoniques sillonneraient les cieux sans aucun souci des passagers, les amerrissages forcés seraient quotidiens, les consignes de sécurité à bord seraient données dans une langue que personne ne comprend, et les passagers seraient débarqués une fois sur deux dans un aéroport qui n'était pas celui de leur destination.

Et pour qui a vu les choses de l'intérieur, et veut les décrire telles qu'elles sont, l'analogie va beaucoup plus loin. Nos avions sont du dernier cri, mais ils intègrent des pièces conçues ou fabriquées dix ou vingt ans auparavant, parfois récupérées sur des épaves. Ils comportent des hublots, mais les passagers sont obligés de se contorsionner pour y voir à travers. Les sièges sont très confortables, mais ne sont pas attachés au plancher. Ces avions ne sont entretenus que lorsque la catastrophe menace, ou lorsqu'elle a déjà eu lieu. Les pièces détachées d'un appareil ne s'adaptent pas sur un autre appareil, et la communication avec les tours de contrôle se fait en une langue difficile que personne ne maîtrise.

Tout au long de cet ouvrage, nous apporterons de telles analogies, afin de permettre au lecteur non-spécialiste de suivre un discours qui se rapporte à cette discipline jeune et fort complexe qu'est l'informatique. Cependant, le but de

l'ouvrage n'est pas de faire des analogies, activité gratuite par ailleurs, mais d'expliquer pourquoi ces « avions » ne volent pas, ou volent mal, pourquoi les passagers sont parfois malades du voyage, et surtout, ce que le passager, la compagnie aérienne, le pilote, la tour de contrôle et le constructeur de l'avion peuvent faire ensemble pour rendre le voyage plus agréable et plus sûr.

Immaturité

Malgré d'énormes progrès récents, les méthodes de conception, les techniques de développement de logiciel, les concepts fondamentaux eux-mêmes, sans parler des moyens mis en œuvre, souffrent tous d'*immaturité*.

Immaturité des processus. On ne s'étendra pas ici sur le modèle à cinq niveaux de maturité des processus de développement (CMM)[1]. Rappelons simplement que la majorité des entreprises en est encore au niveau 1, c'est à dire que le développement se fait chez elles de manière chaotique, inefficace et incontrôlée. Cette situation s'améliore progressivement, mais l'immaturité perdurera, très probablement, pour de nombreuses années. L'évolution est lente, car, pour que les processus changent, il faut que les mentalités changent. Le frein est culturel et humain.

Immaturité des méthodes. On s'accorde tous pour dire que les méthodes ne sont pas parfaites mais qu'elles ont « le mérite d'exister ». Cet argument est pervers. Dans l'industrie, lorsqu'une méthode fabrication est inefficace, on la change ou on l'améliore. La méthodologie devrait être un processus continu, et non un ensemble figé de normes. Mais aujourd'hui la méthodologie est faite sous stress. Les techniques évoluent très vite, et nous voulons que les méthodes évoluent au même rythme. Un exemple significatif : trois gourous se sont réunis pour mettre au point LA méthode de conception par objets[2], qui à l'époque s'intitulait « méthode unifiée ». Il en a découlé, non pas une méthode, mais une notation, un « langage unifié de modélisation » (UML). C'est un progrès, mais moins ambitieux que l'objectif de départ. De fait, la notation a été normalisée avant les concepts, ce qui ne va pas sans problèmes.

Immaturité des concepts, qui sous-tendent les méthodes. Pour reprendre l'exemple d'UML, la notation laisse entrevoir un certain nombre de concepts. Sont-ils clairement définis ? chaque nouveau terme est-il précisé ? si les mots « béton » et « armature » était aussi vaguement décrits que le mot « agrégation » et « association » dans UML, les tours de la Défense ne tiendraient pas.

---

[1] CMM : *Capability Maturity Model*. Indicateur de la maturité des processus de développement, mis au point au Software Engineering Institute, Université Carnegie Mellon.

[2] Tous les termes techniques ou appartenant au jargon informatique sont définis dans le glossaire à la fin de cet ouvrage.

Immaturité des moyens, et en particulier des outils qui supportent les méthodes. Nous ne nous étendrons pas sur cette problématique, abordée dans un précédent ouvrage [CON 98]. Rappelons simplement que dans un contexte hautement compétitif, où techniques et méthodes évoluent à un rythme accéléré, les outils qui supportent la construction du logiciel sont mis à très dure épreuve, et que leurs éditeurs doivent nécessairement faire des arbitrages douloureux. Les outils devraient être pensés de façon cohérente et globale, en tenant compte de l'ensemble des activités de construction du logiciel. C'est rarement le cas. La course-poursuite des éditeurs d'outils les empêche de prendre du recul.

Immaturité des produits, en particulier des applications, mais également de très nombreux progiciels. C'est, la conséquence des points précédents, mais non exclusivement. La qualité des processus est une condition nécessaire, mais non suffisante, à la qualité des produits qui en résultent. En effet, un processus parfaitement au point peut donner lieu à des produits inutilisables. En particulier, aucun processus ne peut garantir que les besoins des utilisateurs ont été bien compris.

Lorsque nous parlons d'immaturité, nous ne jetons la pierre à personne. L'informatique est une activité jeune. Il est naturel qu'elle soit immature à ses débuts. La situation évolue, sans doute moins vite que l'utilisateur ne le souhaite, mais l'évolution est réelle. D'où la question suivante : comment l'utilisateur peut-il apporter sa pierre à l'édifice ? comment peut-il contribuer à l'évolution des produits vers plus de maturité ? car c'est le produit qui intéresse l'utilisateur, et non les processus, les concepts, les méthodes employées.

Immaturité de la vision utilisateur. L'utilisateur est encore trop souvent considéré comme le passager à qui l'on peut cacher les paramètres de vol, et qui ne pourra regarder le paysage qu'à l'arrivée. Et, même lorsqu'il le souhaite, l'utilisateur ou le maître d'ouvrage n'a pas les moyens d'avoir un regard sur les paramètres du projet et sur la qualité du produit. Cependant, comme nous le montrerons dans les chapitres qui suivent, l'utilisateur a dès aujourd'hui la possibilité de connaître, et même d'influencer, la qualité du produit livré. C'est précisément l'objectif de cet ouvrage.

Guide de lecture

Comme tous les ouvrages, celui-ci gagne à être lu linéairement, du premier au dernier chapitre. Cependant, il est possible à un lecteur pressé, ou soucieux de donner la priorité à certains aspects de la valeur du logiciel, d'omettre certains chapitres, quitte à y revenir par la suite s'il le souhaite. Cet ouvrage faisant à

plusieurs reprises l'éloge de la modularité, nous avons essayé de le construire, lui aussi, de manière modulaire.

Le premier chapitre, intitulé *quelques propriétés du logiciel*, traite de sa complexité, de ses propriétés particulières, et de sa maîtrise. Il se termine par quelques conseils pratiques sur le *make or buy* : faut il acheter un logiciel ou le faire développer ? Le traitement de cette problématique donne un avant-goût de la méthode à suivre pour juger rationnellement de la valeur d'un logiciel, et qui sera proposée tout au long de cet ouvrage.

Le second chapitre, *un logiciel pour l'utilisateur*, décrit le rapport triangulaire entre celui-ci, ceux qui le construisent, et ceux qui l'utilisent. Y sont rapidement décrites les approches participatives de la construction du logiciel. Dès ce chapitre, le lecteur-utilisateur peut voir comment, tout en gardant son rôle, il peut être acteur de la valeur ajoutée du logiciel.

Dans le chapitre trois, *évaluer le logiciel*, nous introduisons les techniques encore méconnues et insuffisamment utilisées de l'estimation du logiciel et de sa mesure, et nous lançons une réflexion sur l'utilité d'évaluer le produit logiciel fini *versus* le processus qui a mené à son élaboration. Nous introduisons enfin ce qui sera la matière des six chapitres suivants, à savoir les six caractéristiques qui font la valeur d'un logiciel, quel qu'il soit.

Les six chapitres suivants (ch. 4 à 9), ont la même structure, et traitent des six attributs (ou caractéristiques) qui permettent de qualifier un logiciel et de quantifier sa valeur. Si nécessaire, ils peuvent être lus dans un ordre différent que celui de l'ouvrage. Notons cependant que les deux derniers de ces chapitres, qui traitent de la maintenabilité et de la portabilité, sont plus techniques que les quatre premiers, et quelques notions techniques ont dû y être introduites.

Le chapitre 10 fait une rapide synthèse et montre comment ses six caractéristiques de la valeur du logiciel peuvent être mis en application, à la fois par le maître d'œuvre et le maître d'ouvrage. Il est suivi d'une conclusion, et de plusieurs annexes : dix conseils pratiques, et, nous l'espérons, directement utilisables par les maîtres d'ouvrage, un exemple de proposition d'évaluation de logiciel, un glossaire, des références bibliographiques, et une liste des sites web utiles.

Nous espérons qu'arrivés à la fin de cet ouvrage, le Maître d'Œuvre, le Maître d'Ouvrage et l'Utilisateur de l'informatique auront suffisamment d'outils et de vocabulaire commun pour engager un dialogue constructif, afin d'atteindre ensemble un même objectif : améliorer la valeur de leur logiciel.