Le projet Louvois

Comment maîtriser la complexité informatique ?

Michel Vanden Bossche-Marquette*

Le 15 décembre 2013

L'annonce de l'abandon de « Louvois » par le Ministre Jean-Yves Le Drian doit susciter, au-delà du débat sur les responsabilités, deux réactions :

- Comment se fait-il que la construction d'un tel système d'information soit aussi difficile ?
- Que convient-il de faire pour construire, le plus rapidement possible, le « nouveau Louvois » ?

1. Le secteur public n'est pas le seul à connaître l'échec informatique

Il faut d'abord ne pas céder à la tentation de recourir à des explications par trop simplistes, comme le fait, dans le cas de Louvois, qu'il s'agisse d'un projet mené par le secteur public.

S'il est vrai que de tels projets connaissent de graves échecs – le plus dramatique étant probablement celui du Service National de Santé du Royaume-Uni[1] – les échecs informatiques du secteur privé sont tout aussi nombreux, mais ils restent la plupart du temps inconnus du grand public, fournisseurs et clients faisant preuve d'une grande discrétion ; les difficultés des projets publics sont par contre largement commentées.

2. Un problème logiciel

Alors que matériels et réseaux sont toujours meilleurs et moins chers, le développement logiciel est, depuis l'origine, une source constante de difficultés.

Cette apparente contradiction rend plus obscure encore, au grand public, aux chefs d'entreprise ou au Ministre de tutelle, les causes profondes des difficultés rencontrées dans de nombreux projets informatiques, d'autant que certains d'entre eux ou leurs enfants s'essayent au développement de petites applications avec plus ou moins de succès et que tous utilisent, en privé, de nombreuses applications qui semblent fonctionner raisonnablement bien.

Les études consacrées au taux de réussite des projets informatiques n'apportent pas plus de clarté : évitant d'analyser les causes profondes de la *crise du logiciel*, elles formulent des recommandations évidentes et élémentaires : avoir l'appui de la direction, impliquer les utilisateurs, disposer d'une formulation claire des besoins, planifier l'exécution de manière optimale, etc.[2] Ce sont incontestablement des conditions nécessaires de réussite, mais elles ne sont en aucun cas suffisantes, quand elles ne sont pas irréalistes.

Ainsi, la formulation claire des besoins ne présente pas de difficultés quand il s'agit de mettre en orbite un satellite : on peut recourir à l'arsenal de la physique pour formuler avec précision la nature du problème. Il en va tout autrement pour une application de gestion : la formulation des besoins est très rarement précise et complète. Experts métiers et informaticiens dépensent énormément de temps, d'énergie et d'argent pour se comprendre : c'est le problème de la « faille » entre les experts du métier et les informaticiens (IT), entre maîtrise d'ouvrage (MOA) et maîtrise d'œuvre (MOE).

Reprocher aux utilisateurs une formulation incomplète et vague des besoins est ignorer une évidence : c'est aux informaticiens d'apporter l'outillage conceptuel et technique qui permettra, au métier et à l'IT, d'exprimer, valider et partager une compréhension précise du problème, sans laquelle le développement informatique est voué à l'échec.

De même, s'irriter de voir le métier changer la définition du problème est méconnaître le caractère évolutif de son environnement, d'autant qu'un projet de trop longue durée rendra le changement plus probable et que l'application, une fois livrée, devra être adaptée à de nouvelles exigences.

C'est donc aux informaticiens d'apporter les méthodes et outils permettant de surmonter ces deux difficultés. De son côté, la MOA doit assurer un cadre contractuel qui permette à la MOE de faire preuve de souplesse ; la grande rigidité des marchés publics est en fait la seule difficulté spécifique aux projets publics.

3. Pourquoi la construction d'un logiciel est-elle difficile ?

L'énorme complexité combinatoire des logiciels est la première difficulté à vaincre. Pour construire une application informatique, il faut ajouter à la description du problème métier (qui doit être précise et complète)

[1] http://en.wikipedia.org/wiki/NHS_Connecting_for_Health

[2] http://versionone.com/assets/img/files/CHAOSManifesto2012.pdf

de très nombreux détails techniques indispensables pour que cette description soit exécutable par une « machine » : l'ordinateur, son système d'exploitation, le *middleware* associé, etc. Avec les technologies utilisées et recommandées par la très grande majorité des entreprises informatiques, cette programmation nécessite des centaines de milliers de lignes de code (COBOL, Java, etc.), et souvent plusieurs millions[3]. La « machine » qui en résulte possède en son cœur un système logique à la combinatoire explosive qui la rend profondément instable.

La seconde difficulté est qu'un système tel que Louvois, comme la plupart des logiciels stratégiques, n'est pas une variante d'un système dont la conception et la construction ont été répétées des milliers de fois au cours de l'histoire : c'est un exemplaire unique qui peut, tout au plus, faire usage de quelques « patrons de conception » techniques, sans rapport avec le métier.

Les systèmes classiques – non informatiques, tel un pont – sont très différents. Ils sont stables dans des plages de valeur souvent importantes. Les tests peuvent être limités à des cas de référence à partir desquels il est possible d'interpoler le comportement global du système grâce à la nature continue des lois physiques qui les décrivent. De plus, la construction répétée du même type d'objet sur une longue période a permis de cataloguer et transmettre l'expérience accumulée. Enfin, les tests de ces systèmes sont aujourd'hui la confirmation d'une modélisation a priori, basée sur l'utilisation de théories physiques (dans le cas du pont, la mécanique classique), de leur expression mathématique (le calcul intégral et différentiel) et du recours à des modèles numériques qui rendent le calcul possible à la précision requise (la modélisation par éléments finis).

Il en va tout autrement de la construction des grands logiciels d'entreprise : chaque exemplaire est unique, il n'y a pas de modélisation « mathématique » a priori (sauf pour les systèmes temps-réel hautement critiques) et les tests exhaustifs sont impossibles, le nombre d'états dépassant souvent le nombre de secondes écoulées depuis le *Big Bang* (1 suivi de 18 zéros !).

Ceci explique le taux d'échec important des grands projets informatiques, les problèmes de conception et de complexité[4], et les nombreux *bugs* qui affectent la plupart des logiciels.

4. La complexité dans Louvois

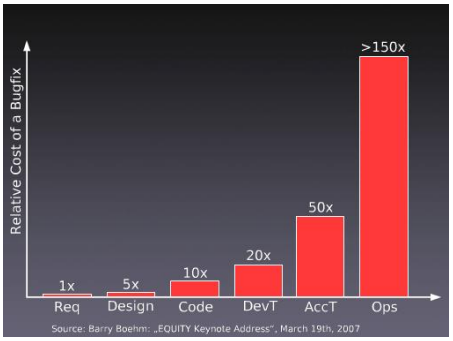
Comme tout logiciel critique, Louvois doit faire face à deux types de complexités :

- La complexité du **« quoi »** : celle du problème à informatiser. C'est une complexité « **essentielle** », liée à l'essence du problème et du projet. Elle est irréductible puisqu'elle résulte ici de la législation et des contrats passés avec les militaires.
- la complexité du « **comment** », celle qui résulte des technologies logicielles utilisées. C'est une complexité « **accidentelle** » puisqu'elle est propre aux moyens techniques choisis, avec une liberté de choix contrainte par l'offre.

Il existe une troisième source de complexité : celle de l'**intégration** avec les systèmes en amont (les SIRH, Systèmes d'Information en Ressources Humaines, à l'origine des événements de paie) et avec les systèmes en aval (éditique, impôts, finances, etc.). Il s'agit, là aussi, de faire la différence entre le « quoi » (quelle est la sémantique de ce que l'on doit intégrer ?) et le « comment » (quels seront les messages, protocoles et architecture utilisés ?).

Dans le développement de Louvois, et dans la pratique courante, la MOA fournit à la MOE une description informelle (sous forme de textes, tableaux et schémas) de ce que le logiciel à construire doit automatiser.

Cette description est interprétée par les informaticiens et est traduite en programmes et structures de données. La « connaissance » métier est donc codée avec un langage de programmation : les informaticiens sont ainsi exposés à une **complexité au carré** : celle du problème (essentielle) et celle de la solution technique (accidentelle). Comme la définition précise et complète du problème est difficile, elle va évoluer dans le temps et entraîner des changements de très nombreuses lignes de code, avec des effets que la combinatoire rend imprévisibles, difficiles à tester et corriger, métier et technique étant totalement imbriqués. L'accumulation des changements augmente l'entropie du code déjà écrit ce qui rend les changements de plus en plus difficiles. Ceci explique qu'une erreur identifiée durant les tests de recette est **50 fois plus coûteuse** qu'une erreur identifiée lors de la spécification[5].

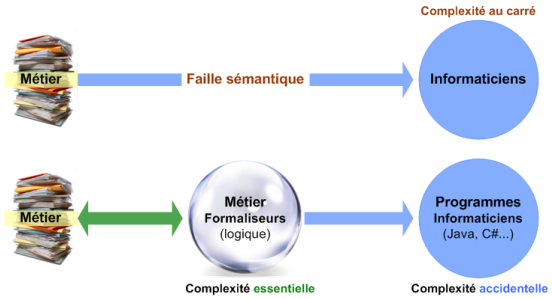


[3] Voir *L'informatique malade des lignes de code*, Sciences & Vie, février 2011
[4] Voir *L'Infotechnocratie, le déni de complexité*, Léon Lévy-Bencheton (2012)
[5] Barry Boehm, *ËQUITY Keynote Address*, March 19th, 2007

Dans Louvois, cette « **complexité au carré** », métier et informatique, est probablement maximale au niveau du calculateur de solde, le cœur du système et la première source des difficultés rencontrées.

5. Que faut-il faire ?

Il faut substituer à la « complexité au carré » deux complexités moindres gérables séparément. Il faut donc **séparer** clairement métier et informatique :



Pour que cette séparation soit effective, la spécification du « quoi » doit couvrir la **totalité** des aspects métier (les concepts, règles et processus métiers).

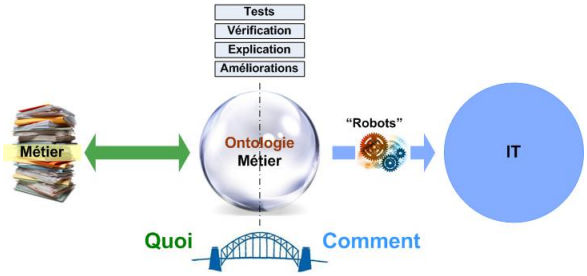
Cette spécification doit être **déclarative** : n'exprimer que de la **connaissance métier** en évitant toute référence au « comment ».

Cette spécification doit être précise et **testable** avant l'écriture de la première ligne du « comment ». Durant les tests, il faut pouvoir **expliquer** le pourquoi des résultats observés car seule une compréhension en profondeur du problème et de sa spécification peut conduire au « **0 défauts** ».

La spécification est un **modèle** qui exprime 100 % de la logique métier. Il doit **piloter l'application** pour que les informaticiens puissent ajouter ce qui n'est pas dans le modèle (et qui ne doit pas y être) sans se soucier de la qualité et de la complexité de la définition métier.

Enfin, comme pour les processus industriels modernes, il faut **automatiser**[6] la construction de l'application.

Le cahier des charges ci-dessus peut être réalisé grâce aux **ontologies** et aux autres technologies du **Web sémantique** du W3C (*World Wide Web Consortium*) :



[6] On évoque ici la génération automatique de code, à ne pas assimiler au *taylorisme* qui n'est pas applicable au développement logiciel (voir note 4)

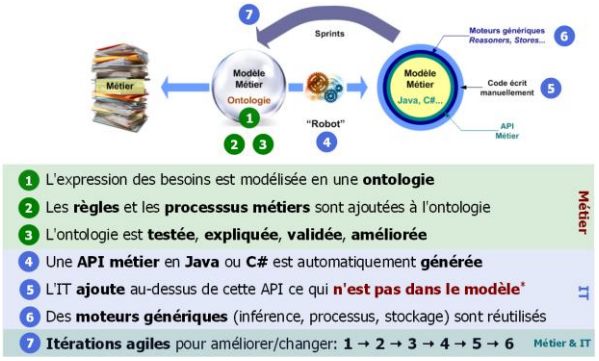
Une **ontologie**[7] est une représentation formalisée des connaissances d'un domaine, partagée par les experts de ce domaine et exploitable par des programmes. Basée sur la logique formelle, elle permet de tenir des raisonnements sur ces connaissances. Si XML exprime une syntaxe, les langages ontologiques (des extensions d'XML) expriment de la sémantique.

6. Quelle application à Louvois, avec quelle conséquence ?

Dans le cas de Louvois, une ontologie serait construite pour modéliser, en priorité, le **calcul des soldes**. Elle définirait les concepts du domaine (militaire, grade, états de services…), les règles (de calcul de solde, de validation, d'inférence…) et les processus associés, sur base des documentations et codes existants. Le calcul des soldes serait le résultat d'un raisonnement générique sur le modèle et les données.

Cette ontologie pourrait être testée avec les jeux d'essai provenant de Condor et des différents SIHR.

Le processus serait itératif : chaque itération fournirait une ontologie validée (après tests et explications) aux informaticiens qui l'exploiteraient, par exemple sous forme de services, dans l'architecture technique actuelle ou adaptée.



On passerait ainsi d'une démarche artisanale (construire manuellement un logiciel comme les Romains construisaient un pont) à une démarche d'ingénierie scientifique (« calculer » un logiciel comme on calcule un pont avant de le construire).

Si cette démarche est encore très largement ignorée dans la pratique quotidienne, elle n'est plus du domaine de la recherche : elle est mise en œuvre avec succès, pour des applications critiques d'entreprise. Elle constitue une véritable innovation capable de relever le défi du « nouveau Louvois ».

[7] http://fr.wikipedia.org/wiki/Ontologie_(informatique)

[*] Michel Vanden Bossche-Marquette est administrateur délégué de Mission Critical IT, une société active dans le domaine des applications pilotées par des ontologies qui exploitent sa plate-forme sémantique ODASE.