De l'évolution de « maintenance »

par Ivan Maffezzini

L'abstraction, intrinsèque au développement du langage, a ses dangers. Elle éloigne des réalités du monde immédiat. En absence d'équilibre on se retrouve avec les futilités des gens à l'esprit vif. A. N. Whitehead

La maintenance du logiciel est un champ du génie logiciel mal considéré et mal compris. T. M. Pigoski.

Si le sens de la réalité existe, et personne peut douter que son existence soit justifiée, alors quelque chose d'autre, que nous appellerons sens de la possibilité, doit aussi exister. R. Musil

Introduction

Au début du XIXe siècle l'analogie avec le système solaire permit à Rutherford de faire avancer la physique atomique. Quand, après la découverte des quanta, elle devint un frein à la compréhension de la structure de la matière, on dut abandonner cette analogie, pourtant si parlante : les physiciens durent alors en trouver de nouvelles et inventer de nouveaux concepts[1] pour « suivre à la trace » la structure des atomes. Cette célèbre révolution scientifique est un très bel exemple du fait que la force du raisonnement analogique, fondée sur la ressemblance partielle avec ce qui est déjà connu, devient une faiblesse devant une « vraie » nouveauté. Le modèle planétaire pour la physique peut être considéré comme l'équivalent des génies traditionnels pour le génie logiciel (GL). L'analogie avec les génies traditionnels a permis de grands pas en avant dans l'automatisation au début du GL, car il fallait consolider une conceptualisation mouvante et difficile à saisir en raison du dynamisme et du désordre propres à ce nouveau champ du savoir. Mais actuellement, l'analogie avec les génies traditionnels est plutôt un frein à la maturation de la discipline, surtout quand on la prend trop au pied de la lettre [1].

Une autre analogie frappa l'imagination de bien des informaticiens dans les années 1970 : celle de la maintenance comme la partie cachée de l'« iceberg logiciel ». Cette analogie faisait suite aux résultats d'études montrant que les coûts après livraison étaient supérieurs aux coûts de développement, équivalant, parfois, même au triple de ceux-ci. Comment une telle situation était-elle possible ? Le logiciel fonctionnait très bien, on avait fait une énorme quantité de tests, le client était satisfait, il avait accepté la livraison… et pourtant. Et pourtant, après avoir livré le produit et avoir commencé à s'en servir dans l'environnement final, il y avait de plus en plus de demandes de changements : erreurs à corriger, adaptation à de nouvelles versions du système

[1] Cette fois, malheureusement, sans analogie avec le monde macroscopique ; ce qui fit dire à R. Feymann que celui qui déclare comprendre la mécanique quantique… n'a rien compris.

d'exploitation, nouvelles exigences des clients ou des utilisateurs… Il y avait assez d'éléments pour se pencher, en adoptant une approche rationnelle — pour ne pas dire scientifique —, sur les problèmes de l'après-livraison, de la maintenance, comme on disait dans les autres génies.

Mais une approche rationnelle demande minimalement de savoir de quoi on parle. Voilà donc la question préalable à toute argumentation sur le statut de la maintenance : « qu'est-ce que la maintenance ? ». Nous allons chercher une réponse partielle dans la constellation conceptuelle informelle proposée par le *Guide to the Software Engineering Body of Knowledge* SWEBOK [2], constellation qui est cohérente avec pratiquement toute la normalisation de l'*Institute of Electrical and Electronics Engineers* (IEEE) et de l'Organisation International de Standardisation (ISO)[2]. Pour cette définition nous n'avons pas estimé nécessaire de considérer certaines ontologies comme celle qui est proposée par Kitchenham et alii. [3] ou celle de Ruiz et alii. [4] car les détails de ces conceptualisations ne contribuent pas à éclaircir le problème que nous abordons ici : « L'emploi du terme "maintenance", emprunté aux machines matérielles, est-il devenu un frein à l'amélioration du GL ? Si oui, faut-il l'abandonner en espérant qu'avec le terme, les approches improductives qu'il sous-tendait disparaissent aussi ? »

Déjà dans la définition du génie logiciel d'IEEE [5] la maintenance acquiert un statut comparable à celui qu'elle a dans les génies traditionnels : « *L'application d'une approche systématique, maîtrisée, quantifiable au développement, à l'exploitation et à la* ***maintenance*** *du logiciel* ». Dans la norme IEEE [6] la maintenance est définie comme : « *Modification d'un produit logiciel après livraison pour corriger les défauts, améliorer les performances ou autres attributs ou pour adapter le produit à un environnement modifié*. » Le syntagme clef pour la contextualisation de cette définition est sans doute « après livraison ». Pour permettre de mieux définir la maintenance, celle-ci est structurée en :

1. *Maintenance adaptative* : modification d'un produit logiciel réalisée après la livraison pour garder un programme informatique fonctionnel dans un environnement changé ou changeant.
2. *Maintenance corrective* : modification d'un produit logiciel pour corriger les défauts découverts après la livraison.
3. *Maintenance perfective* : modification d'un produit logiciel après la livraison pour améliorer les performances ou la facilité d'entretien [6].

Cette structuration est encore aujourd'hui la plus universellement acceptée, même si on parle parfois de *maintenance préventive* (correction des erreurs avant qu'elles ne se transforment en défauts) et d'une sous-catégorie de la maintenance corrective, la *maintenance d'urgence* (intervention imprévue mais nécessaire pour maintenir le système en opération).

---

[2] Ce choix ne nous semble pas restrictif dans la cadre des « Lieux communs » car il s'agit de la définition pratiquement acceptée par tous les praticiens et les théoriciens du GL.

Honneur

Puisqu'il n'existe pas de machines qui n'aient pas besoin d'entretien et puisque plus une machine est complexe et plus elle exige une approche fondée sur la science, la maintenance est indissociable de l'ingénierie. Malgré son originalité, une partie du logiciel aussi est une machine. En dépit des critiques et des doutes sur le statut de la maintenance et malgré le fait que « *la maintenance du logiciel soit* [en 1997] *un champ du génie logiciel mal considéré et mal compris* » [7], depuis quelques années, celle-ci a acquis une grande autonomie organisationnelle et technique et est devenue ainsi un domaine incontesté avec ses connaissances spécifiques, ses outils et son savoir-faire.

Pour ceux qui aiment les fuites en avant et voudraient abandonner le terme « maintenance », il est sans doute utile de relire ce que G. Parikh écrivait il y a plus de vingt ans. Après avoir défini la maintenance pour le logiciel comme « *tout travail fait sur un système en opération, à n'importe quel moment, pour n'importe quelle raison* », il ajoute que « *le terme "maintenance" est un mauvais terme* ». Mais avec un grand sens pratique il conclut ainsi : « *malheureusement dans le logiciel le mot "maintenance" est avec nous depuis longtemps et il semble destiné à y rester* » [8]. Nous croyons qu'il ne faut pas renoncer à ce sens pratique si l'on ne veut pas sortir du domaine du génie, domaine où les mots, loin d'être autonomes, ne sont qu'une simple interface pour mieux saisir et contrôler la réalité naturelle ou celle des machines.

Il ne faut pas renoncer, même si bien des études empiriques citées dans la littérature [7,9] montrent que la maintenance corrective, en moyenne, ne représente que 20 % du coût total de la maintenance. C'est bien parce que le logiciel est fondamentalement différent du matériel que la maintenance autre que corrective est si importante. Ceux qui, en raison des coûts de la maintenance perfective et adaptative, proposent d'abandonner le terme « maintenance » donnent trop d'importance à l'analogie avec le matériel. Malheureusement, parfois, les tentatives de rompre avec l'analogie matérielle créent plus de problèmes qu'elles n'en résolvent : « *Naturellement cette vision de la maintenance* [analogue à la maintenance matérielle] *ne s'applique pas au logiciel, parce que le logiciel ne se détériore pas avec l'usage et le passage du temps* » [10]. Ce constat implique de considérer le logiciel comme un simple programme exécutable, ce qui est loin d'être le cas dans le GL. Si l'on considère le logiciel en tant que produit constitué de plusieurs artefacts, alors il peut se détériorer avec le passage du temps et avec l'usage. Le passage du temps peut le rendre obsolète ou incapable d'inter-opérer avec d'autres logiciels qui ont évolué de manière indépendante et l'usage (l'usage des ingénieurs du logiciel) peut introduire des erreurs dans les artefacts et des incohérences parmi les artefacts. Parler des ingénieurs du logiciel comme des « utilisateurs », même si ce n'est pas orthodoxe, est loin d'être incongru quand on ne considère pas le logiciel comme un simple programme exécutable[3].

[3] À ceux qui sont tentés d'affirmer que le fait de considérer les ingénieurs du logiciel des utilisateurs comme les utilisateurs, que sais-je ?, d'une voiture, est un tour de passe-passe qui n'a d'autre utilité que de défendre la démarche de l'auteur, nous aimerions rétorquer que c'est le fondement même du logiciel qui doit nous porter à voir différemment même le concept d'utilisateur. Pour terminer cette note avec ce qui, injustement, pourrait apparaître comme une provocation, nous ajouterons que les utilisateurs du logiciel tels que considérés normalement dans la littérature ne sont pas les utilisateurs du logiciel mais les utilisateurs du matériel qui exécutent un parmi les n artefacts du logiciel.

C’est surtout la maintenance perfective qui pose problème lorsque l’on parle du statut de la maintenance. Mais quoi de plus normal qu’il existe des demandes de changements au logiciel : la caractéristique fondamentale du logiciel n’est-elle pas la facilité d’adaptation ? La facilité de modification n’est-elle pas ce qui différencie le logiciel du matériel ? Quoi de plus facile que de satisfaire ces demandes, donc ?

En théorie.

En pratique les choses vont tout autrement : bien souvent les changements empirent la situation : dès qu’un changement est effectué d’autres surgissent ; les coûts ne cessent de grimper ; les utilisateurs se plaignent toujours plus ; l’insatisfaction des clients atteint un seuil dangereux… C’est l’élément principal de ce que pendant des années on a appelé la « crise du logiciel ». Ces problèmes de pratique sont assez importants pour que la maintenance devienne un domaine à part entière et que, dans bien des entreprises, elle soit gérée par des structures partiellement autonomes.

Puisque la livraison est ce qui sépare le développement de la maintenance [6], dans le tableau suivant, nous montrons quelques différences, à notre avis significatives, mais qui n’ont aucune prétention d’exhaustivité, entre les activités concernant les phases de pré-livraison et celle de post-livraison. Pour suivre la structuration de la maintenance présentée dans l’introduction, le tableau a été divisé en trois parties :

- correction des erreurs
- adaptation à un nouvel environnement
- mise en œuvre de nouvelles exigences (fonctionnelles ou de qualité)

Table 1 Comparaison entre pré et post-livraison

		Pré-livraison	**Post-livraison**
Correction des erreurs	Informer le personnel technique.	Facile. Souvent non officielle, avec tous les dangers du non officiel (oublis, difficultés de suivi, etc.).	Une procédure officielle et détaillée peut créer des retards et de la frustration. Contraintes de temps beaucoup plus fortes. Lorsqu'on évite la procédure officielle pour gagner du temps, on augmente les risques d'erreurs plus que dans la pré-livraison (configuration).
	Analyser	L'activité peut s'intégrer facilement dans les activités normales. Contraintes organisationnelles peu importantes.	Contraintes organisationnelles fortes. Souvent longue, elle requiert parfois l'intervention des développeurs pour bien comprendre le logiciel car souvent la documentation n'est pas axée sur l'entretien.
	Corriger	Ce sont surtout les capacités et les connaissances du système du correcteur qui sont en jeu. La documentation est relativement importante.	Facilité de créer des effets de répercussion en chaîne (« Ripple) à cause du manque de connaissance « pratique » des points faibles du système.
	Tester	Moins de pression temporelle. Par contre, dans les organisations pas assez mûres, les tests peuvent être pilotés par les développeurs.	Harcèlement fréquent du client et des utilisateurs. Tests pilotés par les échéances.
	Installer	Pas de problèmes particuliers car le système n'est pas en service.	Doit toujours être fait « vite et bien » en dérangeant le moins possible (parfois pas du tout) le fonctionnement normal.
Adaptation à un nouvel environnement	Étudier l'environnement	Peut être relativement facile à cause des bonnes connaissances techniques et de l'historique de l'environnement précédent.	Peut être très long à cause du manque d'expérience technique.
	Changer les interfaces	Intervention dans des programmes bien connus.	Parfois difficile de prévoir les effets de bord.
	Tester	Mêmes considérations que pour la correction des erreurs.	Mêmes considérations que pour la correction des erreurs.
	Installer	Mêmes considérations que pour la correction des erreurs.	Mêmes considérations que pour la correction des erreurs.

		Pré-livraison	**Post-livraison**
Améliorations et ajouts	Accepter les demandes	On peut être plus conservateurs car « il faut avant tout livrer un système fonctionnel ».	Pour « garder » le client il est souvent nécessaire d'accepter toutes les requêtes des utilisateurs.
	« Cycle » de modification	Activités et procédures normales de développement.	Intégrer et adapter les activités et les procédures de développement dans un cadre contraint où la correction des erreurs est normalement prioritaire.
	Installer	Pas de problèmes particuliers, car le système n'est pas en service.	Activité délicate pour ne pas perturber le travail des utilisateurs.

Même si le tableau est loin d'être complet, il nous semble qu'il permet d'affirmer que les contraintes opérationnelles et organisationnelles ont un impact tel sur le projet qu'il serait irréfléchi de considérer le développement initial et la maintenance comme un seul et même processus. La norme ISO 12207 [11], par exemple, considère la maintenance et le développement comme deux processus primaires séparés et crée également un lien d'inclusion entre les deux :

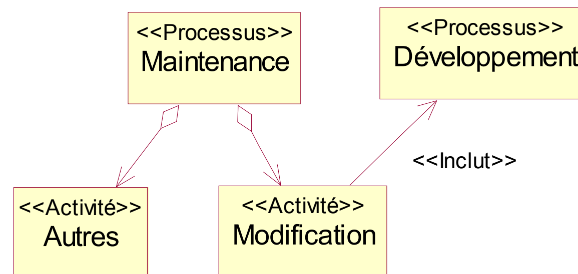

Figure 1 Maintenance versus développement selon ISO 12207.

La figure précédente montre que la maintenance « se sert » du développement lors de l'activité de modification. Mais la norme ne présente pas que ce lien : dans pratiquement toutes les activités du développement, le personnel doit faire une évaluation selon le critère de « *faisabilité de la maintenance* ». Ces évaluations font donc déborder la maintenance dans la phase de pré-livraison. Comme on le précise de façon synthétique dans SWEBOK [2] : « *La phase de maintenance du cycle de vie commence après la livraison, mais les activités de maintenance commencent bien avant* ». La maintenance est donc complètement au centre, ce qui est cohérent avec la vision du logiciel comme ensemble d'artefacts facilement modifiables.

Même si la séparation entre maintenance perfective et corrective est importante du point de vue théorique, en pratique, le besoin de faire des corrections pousse le personnel de maintenance à introduire des améliorations et de nouvelles fonctionnalités, ce qui est fort normal, malgré les dangers d'introduire de nouvelles erreurs, non seulement pour des considérations psychologiques « une fois que l'on a ouvert la boîte pourquoi ne pas… » mais aussi à cause des particularités du logiciel dont l'essence est d'« axer sur la modification ».

Déshonneur

Pour appuyer notre thèse (« la maintenance ne comporte pas d'activités suffisamment particulières pour justifier son existence en tant que domaine séparé ») nous allons commencer par considérer deux sources [4,12] citées dans un article de April et *alii* [9] qui défend la thèse contraire à celle qui est exposée ici, c'est-à-dire l'« *unicité des activités du processus de maintenance* ». Ce dernier texte a été choisi parce qu'il s'inscrit de plein droit dans la ligne du GL telle que prônée par IEEE, par ISO et par le *Software Engineering Institute* (SEI)[4] et à laquelle nous adhérons. Les deux sources sont des articles relativement récents (respectivement de 2004 et de 2001), parus dans *Journal of Software Maintenance: Research and Practice* et qui sont censés jeter les bases permettant de répondre à la question posée lors *du 3rd Annual Workshop on Empirical Studies of Software Maintenance :* « *Quelles sont les différences entre les habilités, les méthodes et les outils de la maintenance et ceux du développement* ? » [3]. Il ne s'agit bien sûr pas de « critiquer » ces articles mais, simplement, de voir si les conclusions des auteurs ne pourraient pas être fort différentes si l'importance de la maintenance (ou de la maintenance *qua* évolution) n'était pas le cadre intouchable dans lequel ils s'inscrivent.

1. *An Ontology for the Management of Software Maintenance Projects* [4]. Nos considérations seront fondées sur les six (6) diagrammes UML employés pour représenter les concepts de la maintenance :

   - *Product ontology diagram*. Ce diagramme s'applique tel quel au développement. Il ne comprend pas de concepts propres à la maintenance.
   - *Activity subontology diagram*. Dans le diagramme suivant nous avons repris la partie qui traite de la maintenance :

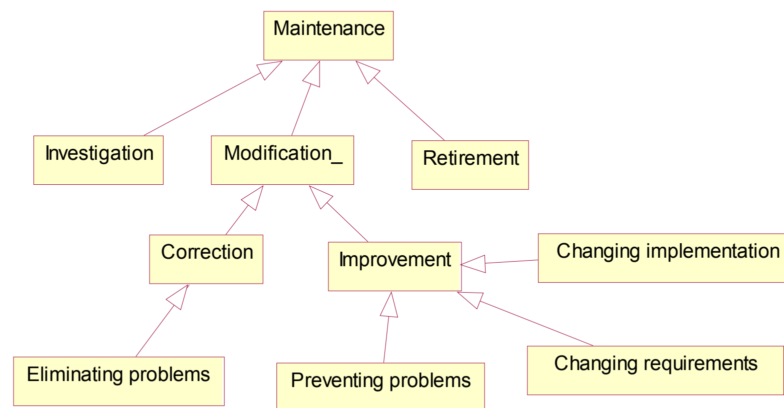

Figure 2: *Activity subontology diagram* [4] (partie maintenance)

   Comme on peut le constater, il ne s'agit que d'une variation de la classification classique. La partie du diagramme que nous n'avons pas représentée s'applique autant au développement qu'à la maintenance.

   - *Process organisation subontology*. Des 34 concepts représentés, un seul est lié à la maintenance « *Maintenance request* ». Ce concept est une généralisation de « *Change request* » et de « *Problem report* » qui font plus

[4] Un des auteurs est *Executive Editor* du *Guide to the Software Engineering Body of Knowledge* SWEBOK.

proprement partie de la gestion de la configuration (processus auxiliaire asservi au développement autant qu'à la maintenance [11]).

- *Measurement ontology* et *Workflow ontology*. Dans ces deux diagrammes il n'y a rien qui puisse concerner la maintenance *qua* maintenance.
- *Agent ontology.* Parmi les organisations, il y a le « *Maintener* », ce qui n'est une nécessité que si, au préalable, on n'avait pas établi que la maintenance doit être un domaine séparé.

2. *Type of Software Evolution and Software Maintenance* [12]. Les auteurs proposent une nouvelle classification car ils jugent, justement, que les classifications existantes ne sont pas assez précises et empêchent de comparer le comparable et donc de faire avancer la maintenance tant du point de vue théorique que du point de vue pratique. Leur approche s'oppose à l'approche fondée sur les intentions qui est à la base de la classification classique (corrective, adaptative, perfective et, éventuellement, préventive). Leur classification des activités est fondée sur « *une évidence objective vérifiable par des observations* ». Chaque activité appartient à un type et à un agglomérat (*clusters*). Un agglomérat peut contenir des activités de plusieurs types. La liste des 158 activités présentée dans l'article est, selon les auteurs, « *loin d'être exhaustive* ».

   - *Types*. Les types possibles sont : formation (*training*[5]), consultation (*consultive*), évaluation (*evaluative*), changement de forme (*reformative*), mise à jour (*updative*), pansement (*groomative*), prévention (*preventive*), performances (*performance*), adaptation (*adaptive*), réduction (*reductive*), correction (*corrective*), croissance (*enhancive*).
   - *Agglomérats*. Pour faciliter la classification les auteurs ont inséré les activités dans quatre (4) agglomérats (*clusters*) : règles d'affaires, propriétés du logiciel, documentation, interfaces de soutien. Si les trois premiers agglomérats n'ont pas besoin d'explications particulières, il faut sans doute ajouter quelques mots sur le quatrième. Pour qu'une activité fasse partie des *interfaces de soutien* il faut que « *la réponse à la question "Les activités changent-elles le logiciel ?", soit un "non" qui provient d'une évidence objective* ».

   À notre connaissance, cette structuration est assez originale et donc nécessite quelques précisions :

   - *Correction* ne s'applique qu'aux règles d'affaires.
   - *Pansement* est ce que l'on appelle normalement *correction*.
   - *Mise à jour* et *changement de forme* ne s'appliquent qu'à la documentation.

Nous estimons que non seulement les deux articles n'apportent pas de soutien à « l'unicité de la maintenance » mais qu'ils sont loin de faciliter la réponse à la question posée au *3rd Annual Workshop* cité plus haut [3]. Dans les deux cas, nous sommes en présence d'une photo de la réalité actuelle considérée du point de vue de quelqu'un qui est concerné par la maintenance et qui donc y trouve… la maintenance bien installée.

[5] Nous avons traduit les adjectifs pas les substantifs pour ne pas rendre le français trop lourd.

Mais, pour répondre à la question de l'unicité, il faut comparer les activités de maintenance avec celles du développement initial. En ce qui concerne l'article sur l'ontologie [4], la comparaison est facile et, comme nous l'avons déjà souligné, il n'y a pas de différences significatives. Pour ce qui est de l'article sur la typologie [12], la comparaison demanderait un long travail préalable de définition des activités de développement, définition qui devrait être construite avec le même niveau de granularité et en respectant les mêmes règles. De plus, pour que la comparaison ait un véritable intérêt pour le praticien, il faudrait mesurer les coûts des activités car, même si on trouvait qu'il y a un nombre élevé d'activités différentes, si ces activités ne « pèsent » pas dans le cycle de vie, il ne faudrait pas les considérer.

Nous terminerons cette brève analyse des deux articles avec un commentaire au propos de G. Parikh cité dans la section précédente. Si, en 1986, G. Parikh avait sans doute raison d'écrire que « *malheureusement dans le logiciel le mot "maintenance" est avec nous depuis longtemps et il semble destiné à y rester* » [8], il nous semble évident qu'aujourd'hui ce destin est, au moins partiellement, entre nos mains et donc… dans ces conditions, il ne s'agit plus d'un destin. En raison des nouvelles méthodes de développement, des nouveaux outils disponibles, des formations universitaires beaucoup plus axées sur la qualité et la productivité — à cause de l'évolution du GL, en peu de mots — la maintenance, comme processus séparé du développement, n'a plus des raisons d'être. À moins que l'on ne considère que l'existence de départements de maintenance, de revues qui traitent exclusivement de maintenance, de cours universitaires sur la maintenance… ne soient pas un motif suffisant pour justifier que la discipline continue à exister.

Mais que se passe-t-il si l'on se déplace du côté des méthodes agiles, de ces méthodes qui vont de *eXtreme Programming* [XP] à *Unified Process* [UP][6] ? Pour ce qui concerne XP peut-on dire quelque chose de plus concis et juste que ce qu'écrivait K. Beck : « *En vérité la maintenance est l'état normal d'un projet XP* » (cité en [13]). Ce qui veut dire que, dans un projet XP, il n'y a pas de différence entre développement et maintenance car tout ce qui est particulier à l'entretien du logiciel est intégré dans l'approche normale de développement. En d'autres termes, dans un projet XP, le mot « maintenance » est inutile car il n'est que synonyme de développement.

Considérons maintenant UP, l'autre extrême du spectre des méthodes agiles. L'intérêt de la méthode réside surtout dans l'approche incrémentielle et itérative [14]. Mais si on livre par incréments, tout au long du cycle de vie du logiciel, pendant qu'on « développe » la partie n, on « maintient » les parties de 1 jusqu'à n-1. Donc, la frontière entre les deux parties devient artificielle, à moins que la structure organisationnelle n'impose une division factice du point de vue des outils, du savoir-faire et des méthodes. Ce qui équivaudrait à dire que la structure organisationnelle est là pour mieux gérer la maintenance qui, elle, n'a d'existence, séparée du développement, que pour satisfaire la structure organisationnelle. Donc, il faut conclure que s'il existe un département de maintenance, son but principal est de « s'inventer » des travaux pour continuer à exister[7].

---

[6] Nous considérons UP comme une méthode agile même si cette inclusion peut soulever des objections solides.

[7] Ce qui pourrait nous faire penser à l'organisation de l'« Action parallèle » dans *L'Homme sans qualité* de R. Musil.

Il est pour nous assez évident qu'il existe des activités d'entretien du logiciel, mais ce qui n'est pas évident du tout, c'est qu'elles doivent être rassemblées sous le terme « maintenance », car ce terme implique une séparation du développement qui n'a pas de fondement objectif sinon l'inertie organisationnelle (des entreprises) et conceptuelle (des chercheurs). Si l'on veut employer le terme « évolution », alors celui-ci doit être pris au pied de la lettre et non comme un synonyme plus ou moins strict de maintenance.

Pour terminer, voici des citations de livres bien connus qui nous semblent être des bons exemples de ce qu'il ne faudrait pas faire :

- « *La maintenance du logiciel a fait partie de ma vie pour plus de vingt ans* » [7]. Cette phrase qui est censée donner une certaine valeur au contenu du livre, est une arme à double tranchant qui permet de répondre : « C'est bien parce que vous avez le nez dans la maintenance depuis vingt ans et vous y avez tellement investi psychologiquement que vous ne pouvez pas avoir assez de recul pour voir que tout cela pourrait n'être qu'une variation légère de quelque chose d'autre. »

- « *L'ingénieur de maintenance doit avoir une plus vaste gamme de compétences qu'un programmeur. Entre autres, il doit avoir une grande facilité de compréhension et une capacité d'analyse très étendue* » [15]. On dit cela des ingénieurs des exigences, des concepteurs, des architectes… On dit cela toutes les fois que l'on a une vision très étroite de la « programmation » (ce qui arrive bien souvent). Ce genre d'affirmation est du même genre que celle-ci : « *Le vrai travail de la maintenance logicielle est de type cognitif et a lieu dans l'esprit des informaticiens impliqués*. » [16] Quand on pense que ces affirmations sont employées pour caractériser la maintenance et qu'elles reflètent la culture dominante du domaine, on ne peut que fredonner : « Faut-il pleurer ? Faut-il en rire ? »

Comme l'écrit M. A. Colter cité en [9] : « *Le plus gros problème de la maintenance du logiciel n'est pas d'ordre technique mais de gestion* ». Et, à nous d'ajouter : et surtout d'ordre organisationnel.

Au-delà

Si la réflexion suivante de T. M. Pigoski « *La maintenance du logiciel est un champ du génie logiciel mal considéré et mal compris. Bien que la maintenance du logiciel existe depuis des années, rares sont les écrits qui en traitent* » [7] pouvait être vraisemblable en 1996, elle ne l'est plus en 2007, à l'heure où les articles et les conférences qui traitent de la maintenance sont si nombreux. Qu'il suffise de dire, à titre d'exemple, qu'un petit livre comme celui de Grub, publié en 2003, contient 300 références [15]. On n'est donc plus à l'étape où il faut « prouver » l'importance de la maintenance. C'est en raison de tout cet enrichissement du domaine et de l'autonomie acquise qu'aujourd'hui on peut se poser la question de savoir si cette autonomie est vraiment utile. Utile dans le sens où elle favorise la maîtrise des processus, ainsi que l'augmentation de la productivité et de la qualité. Se

poser la question ne veut pas dire connaître la réponse, mais tout simplement souligner un malaise possible dans la situation actuelle.

Pour appuyer notre réponse aux questions initiales « L'emploi du terme "maintenance", emprunté aux machines matérielles, est-il devenu un frein à l'amélioration du GL ? Si oui, faut-il l'abandonner en espérant qu'avec le terme, les approches improductives qu'il sous-tendait disparaissent aussi ? » nous allons considérer l'article de N. Chapin et alii [12]. On y propose la définition suivante : « *Application délibérée à du logiciel existant d'activités ou des processus [...] qui modifient soit la façon dont le logiciel dirige le matériel du système soit la façon [...] dont le système contribue aux affaires des parties prenantes* […] *avec les activité et les processus d'assurance de la qualité* […] *le tout souvent appliqué dans le contexte de l'évolution du logiciel*. » Comme les auteurs le soulignent, ils ont laissé tomber le syntagme « après livraison » qui est à la base de la définition d'IEEE [6]. Ce qui veut dire que l'on peut faire de la maintenance pendant tout le cycle de vie du logiciel. La seule différence entre développement et maintenance est « logiciel existant ». Mais, puisque les auteurs définissent le logiciel comme : « *La partie non matérielle, en incluant la documentation associée, d'un système partiellement ou intégralement mis en œuvre avec un ordinateur* […] » dès qu'il y a le moindre document[8], il y a du logiciel et donc on est toujours dans le processus de maintenance. Les auteurs différencient la maintenance de l'évolution dont ils donnent la définition suivante : « *L'application des activités et des processus de maintenance qui génèrent une nouvelle version opérationnelle du système avec des changements, fonctionnels ou de qualité, par rapport à une version antérieure. Les changements doivent être expérimentés par le client. L'intervalle de temps entre les versions peut varier de moins d'une minute à des décennies* […]. *Parfois le terme est employé de manière restrictive comme synonyme de maintenance et parfois de façon plus large pour indiquer la séquence d'états et de transitions par lesquels passe le logiciel de sa création initiale jusqu'au retrait.* » Si on oublie l'interprétation restrictive qui réduit l'évolution à la maintenance, le terme « évolution » couvre tout le cycle de vie du logiciel exactement comme la maintenance. La différence n'est donc pas dans l'extension mais dans le fait qu'il s'agit d'évolution lorsque l'application des activités de maintenance entraîne des changements visibles par le client. Si l'on considère que le logiciel est autre chose que du code (ce dont les auteurs sont conscients), on peut synthétiser la position de l'article comme suit : « *Dès que l'on démarre un projet, on est dans une phase de maintenance et le projet évolue d'une livraison à l'autre en modifiant le fonctionnement de façon perceptible par le client.* » Tout ce qui n'a pas d'impact sur l'exécution ne fait pas partie de l'évolution et fait partie strictement de la maintenance : les agglomérats « documentation » et « interfaces de soutien » et les activités de type *prévention* ou *pansement* lorsqu'il s'agit de l'agglomérat « propriétés du logiciel ». Ce qui fait partie en même temps de l'évolution et de la maintenance, ce sont les activités concernant les règles d'affaires et les activités de type « adaptation » et « performances » de l'agglomérat « propriétés du logiciel ».

L'article que nous venons d'analyser, tout comme la norme ISO 12207 (« *Celui qui fait la maintenance doit suivre le processus de développement pour mettre en œuvre les*

[8] Malheureusement, les auteurs ne sont pas cohérents dans l'emploi du terme documentation qui forme un agglomérat séparé des règles d'affaires et des propriétés du logiciel.

modifications » [11]) et XP (le développement est maintenance) font tomber le clivage entre l'avant et l'après livraison. Mais notre impression est qu'aucun des trois n'ose trop s'éloigner de la conceptualisation actuelle pour chercher de nouvelles voies. Il faudrait arriver à trouver d'autres façons de classer les activités en GL, chercher de nouvelles constellations conceptuelles moins fondées sur l'analogie avec le matériel et qui répondent mieux à l'essence (facilité de modification) du logiciel.

Nous sommes loin de pouvoir en proposer, mais abandonner le terme « maintenance » pourrait être un premier pas dans la recherche de nouvelles classifications. Nous terminons sur une liste désordonnée et hétérogène de possibles points de départ pour analyser les conséquences d'une possible fin de la maintenance *qua* maintenance.

1. En raison des progrès en GL, la rétro-ingénierie et la réingénierie perdront beaucoup d'importance dans les prochaines années, car les systèmes que l'on léguera seront nos systèmes…

2. Le monde sera toujours plus informatisé et donc les interfaces entre les machines seront toujours plus importantes. Ceci pourrait impliquer des exigences plus stables, au moins du point de vue fonctionnel.

3. Le couplage et la cohésion, qui ont tant promis en informatique, pourront-ils être employés pour mieux mesurer les activités et leurs interactions dans le processus d'automatisation (qui ne devrait pas se limiter au logiciel) ?

4. Penser une classification des domaines d'application selon des discriminateurs qui tiennent compte de la stabilité et du niveau d'automatisation. Un domaine « volatile » ne devrait sans doute pas être traité comme un domaine stable.

5. L'existence des départements de maintenance (ou l'externalisation) ne peut pas être considéré comme un simple enjeu de pouvoir.

6. La documentation (les modèles) peut favoriser la compréhension du système existant et en même temps être un frein à la compréhension du monde dans lequel le système opère.

7. Quelle est la valeur de données recueillies il y a vingt ans pour un domaine en changement comme le GL ?

8. Le GL a-t-il besoin d'une révolution copernicienne ? Et si jamais cette révolution advenait, saurons-nous nous rappeler que la vraie révolution copernicienne n'a pas changé le mouvement des planètes ?

9. La fonction des humains sera-t-elle un jour de réparer les logiciels qui ne réussissent pas à s'auto-réparer ?

10. Quand est-il préférable d'avoir une documentation sans code plutôt que du code sans documentation ?

11. Les artefacts qui composent le logiciel sont du matériel.

Pour revenir sur les analogies : un jour sans doute les ingénieurs du logiciel souriront en pensant à nos efforts pour déterminer la place de la maintenance comme aujourd'hui nous sourions en pensant qu'au début de la mécanisation de l'agriculture il y en eut pour proposer de conduire les tracteurs avec des rênes — ce qui n'était sans doute pas dû au manque d'inventivité du concepteur mais plutôt à la peur de voir les agriculteurs se cabrer devant la nouveauté du volant.

Références

[1] I. Maffezzini et alii, « Prolégomènes à une critique du génie logiciel - Partie I : Contextualisation », *Génie Logiciel*, Septembre 2003, Numéro 66.

[2] IEEE computer society, *SWEBOK - Guide to the Software Engineering Body of Knowledge*, 2004.

[3] B. Kitchenham et alii, « Toward an Ontology of Software Maintenance », *Journal of Software Maintenance : Research and Practice*, 11 (6): 365-389.

[4] F. Ruiz et alii, « An Ontology for the Management of Software Maintenance Projects », *Journal of Software Maintenance: Research and Practice*, 14 (3): 323-349.

[5] IEEE Std 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology,* 1990.

[6] IEEE Std 1219-1998, *IEEE Standard for Software Maintenance,* 1998.

[7] T.M., G., Pigoski, *Practical Software Maintenance*, John Wiley & Sons, 1997.

[8] G. Parikh, *Handbook of Software Maintenance*, Wiley and Sons, 1986.

[9] A. April et alii, "Software Maintenance Maturity Model (SMmm): the software maintenance process model", *Journal of Software Maintenance: Research and Practice*, 17: 197-223

[10] G. Canfora, A. Cimitile*, Software Maintenance*, ftp://cs.pitt.edu/chang/handbook/02.pdf (Consulté le 22 juillet 2007)

[11] ISO/IEC std. 12207, *Information Technology – Software Life Cycle Processes*. 1995

[12] N. Chapin et alii, "Type of Software Evolution and Software Maintenance", *Journal of Software Maintenance: Research and Practice*, 13: 3-30

[13] C. Poole and J. W. Huisman, "Using Extreme Programming in a Maintenance Environment", *IEEE Software,* vol. 18, iss. 6, November/December 2001, pp. 42-50.

[14] J. Arlow, I. Neustadt, *UML2 and the Unified Process*, Addison Wesley, 2005.

[15] P. Grub, A. Takang, *Software Maintenance – Concepts and Practice*, World Scientific, 2003.

[16] R. Kusters, F. J. Heemstra, "Software maintenance: an approach towards control", The Open University, The Netherlands