Frictions / créations

Les enjeux de la conception incrémentale

Emmanuel Gaillot (egaillot@octo.com)
Bernard Notarianni (bnotarianni@octo.com)

Résumé

La construction d'applications informatiques génératrices de valeur pour les utilisateurs nécessite la confrontation et la résolution de contraintes fonctionnelles et techniques. Cette confrontation peut se faire au travers de document écrits ou par des échanges oraux, entre des spécialistes du fonctionnel et des spécialistes de la technique, ou même des généralistes à double compétence. Dans tous les cas, cette confrontation engendrera des frictions et potentiellement des frustrations réciproques.

Nous pensons que ces confrontations et les frictions sont les conditions nécessaires (non suffisantes) pour que l'équipe puisse produire une application qui apporte de la valeur aux utilisateurs.

Les pratiques XP peuvent nous aider à traiter efficacement ces frictions en créant le contexte qui permettra de les mettre en valeur et de concentrer l'énergie de l'équipe à les adresser, plutôt qu'à les éviter.

De l'incompréhension au noeud traumatique

Vu de suffisamment loin, la conception logicielle pourrait sembler simple : un problème est posé – *quoi* faire – auquel il « suffit » de trouver la réponse – *comment* faire – pour pouvoir ensuite l'implémenter.

Rapprochons nous du sol et la réalité est tout autre. Des difficultés apparaissent. Chaque demande des utilisateurs entraîne une levée de boucliers : trop difficile à construire, trop coûteux à entretenir, trop lent à exécuter. On peut invoquer la mauvaise foi ; ces récriminations sont pourtant souvent les indices de réelles limitations techniques. Les performances des machines sont insuffisantes. Les fonctionnalités offertes par le système d'exploitation limitées. Les algorithmes à mettre en oeuvre insatisfaisants.

Les informaticiens connaissent ces limitations et savent en expliquer les causes. Ils ont une idée précise des améliorations techniques à apporter, des contournements possibles pour patienter jusqu'à la prochaine version. Chacune de ces idées renvoie à un vocabulaire précis – un jargon technique – qui n'a pas été inventé dans le but d'aider l'utilisateur final, et qui le rend perplexe. Bus, méthodes, information, public, interfaces, process, curseur, table, données… le langage informatique est insidieux, emprunt de doubles sens. Mystères pour les profanes, arcanes pour les initiés.

La faute aux technocrates, alors ? Le problème n'est pas si simple. Les informaticiens sont eux-mêmes dans une position délicate. La manière dont les utilisateurs formulent leurs demandes renvoie à un autre langage technique – le *langage métier*. Situation miroir... les informaticiens sont eux aussi face à un langage étranger qu'ils ne comprennent pas et qui leur échappe : d'autres études, d'autres discussions avec les confrères, d'autres expériences professionnelles.

Ceux qui ont vécu de douloureuses expériences sur un projet informatique ont identifié et anticipent ces heurts de communication entre les spécialistes de la technique et ceux du métier. Consciemment ou inconsciemment, le souvenir de ces douleurs passées et l'anticipation de celles à venir teintent les stratégies utilisées pour construire des logiciels.

Le document, naturellement

Et si les incompréhensions au sein de l'équipe étaient dûes à un manque de communication, un manque de précision ? Tout se passe comme si l'équipe au sens large ne partageait pas assez d'informations... et que celles partagées étaient ambiguës.

Cherchez une solution à ce manque de quantité et de qualité, et le document écrit apparaît comme un candidat tout trouvé. Ses atouts : il peut décrire une information à un nombre illimité de personnes – internes ou externes à l'équipe, actuelles ou futures. Un document peut être facilement copié et diffusé ; sa lecture peut être parallélisée à l'envie. Il promet que tout un chacun pourra avoir le même niveau de perception du projet que les autres lecteurs.

S'agit-il pour autant d'un passage obligé ? Le document décrit une information, le logiciel informatique également. Peu étonnant que le document écrit ait une telle omniprésence dans les métiers de l'informatique. En parfaite logique circulaire, les informaticiens construisent des logiciels – traitement de texte, tableurs, assistant de projet, de dessin, de conception UML, de suivi des anomalies, wiki... – qui leur permettront de rédiger encore davantage de documents pour décrire d'autres logiciels en devenir. Les projets informatiques finissent par se réduire à l'ensemble de la documentation qu'ils génèrent. “Il faut documenter davantage !” Avec autant de marteaux, on finit par voir des clous partout.

Soyons conscients par ailleurs des problèmes qu'induisent les documents écrits. L'information partagée est nécessairement déformée par le point de vue du rédacteur ; elle est sujette à contre-interprétation (le point de vue du lecteur étant différent). Pire encore, il n'y a pas moyen de “vérifier” auprès du document qu'on a bien compris le message qu'il transmettait, contrairement à ce qu'on peut faire lors d'une communication orale.

En fait, on peut se demander si le document n'est pas tant rédigé pour le bénéfice du lecteur que pour *éviter* tout retour d'information (*feedback*). Retour de la part du rédacteur qui pourrait corriger une interprétation incorrecte ou répondre à une critique un peu vive – mais également retour de la part du lecteur, qui pourrait questionner le fond ou la forme du document. Si deux personnes veulent se comprendre, est-il alors seulement possible qu'elles y parviennent par document interposé ?

L'équipe intégrée – la réponse ?

Si la production de documents est devenu le centre des méthodes de développement logiciel traditionnelles, finissant par en devenir le symbole, les méthodes agiles organisent une contre-culture. D'artéfact de communication, le document écrit remplit désormais une fonction d'archivage, d'historisation. L'essentiel de la communication ne se fait plus par écrit, mais par oral, au sein d'une équipe colocalisée.

Les méthodes agiles auraient-elles résolu la question des incompréhensions mutuelles entre spécialistes de la technique et spécialistes du fonctionnel ? Ici, la distance qu'introduisait le document entre l'émetteur et le récepteur n'existe plus. Cette immédiateté n'abolit pas les frustrations pour autant. Les experts métier continuent de hausser les yeux au ciel quand on leur explique que la base de données ne peut pas contenir des dates sur quatre chiffres, les développeurs continuent de penser que la politique de sécurité attendue par les utilisateurs est trop compliquée et inutile. Tout au moins – et c'est déjà là un progrès significatif – l'une et l'autre des deux parties n'ont plus l'illusion d'une compréhension mutuelle.

Les incompréhensions n'en sont pas résolues ; les méthodes agiles ne font que les révéler de manière encore plus cruelle.

La troisième voie ou le fantasme de la double compétence

Puisque langage technique et langage fonctionnel coexistent, la personne idéale pour construire l'application ne serait-elle pas encore quelqu'un qui parle ces deux langages ? L'engouement récurrent des recruteurs informatiques pour les “doubles compétences” suit ce schéma de pensée. Naissance d'un mythe, d'un archétype (appelons-le “le Double Expert”) recherché partout et n'existant nulle part.

On attend du Double Expert de savoir appréhender de manière efficace aussi bien les contraintes fonctionnelles que techniques. Lorsqu'il s'agit de prioriser les fonctionnalités à implémenter, il sait analyser les données du métier et sélectionner ce qui peut être automatisé à moindre coût. Lorsqu'il s'agit à l'inverse de choisir les outils pour développer efficacement l'application, il a le recul nécessaire pour choisir un outil adéquat. Il sait à lui seul orienter le projet selon une double contrainte : minimiser les coûts techniques, maximiser la valeur fonctionnelle.

Quand bien même le Double Expert serait capable d'analyser l'ensemble des contraintes techniques ou fonctionnelles qu'il perçoit, son analyse n'en serait pas moins limitée par son horizon de perception. Le Double Expert n'a pas le niveau d'expertise voulu ? Qu'importe. Créons des équipes de (presque) Doubles Experts. Mais ne s'agit-il pas là d'un retour au problème initial, celui auquel la double compétence devait apporter une réponse ? À réinstaller plusieurs personnes sur un même projet, on multiplie à nouveau le nombre de points de vue possibles, et l'équipe se retrouve confrontée à la difficile conciliation des différentes perceptions des contraintes en jeu.

“L'Enfer, c'est les autres”

Que l'équipe soit constituée de spécialistes de la technique, de spécialistes du fonctionnel ou encore de généralistes bi-compétents exclusivement, chaque membre de cette équipe apporte son expérience propre pour résoudre le système de contraintes qu'il perçoit. Il apporte ainsi une certaine vision des problèmes, et ses solutions, compréhensibles ou non par le reste de l'équipe.

Se profile une clef de compréhension des phénomènes de frictions : les solutions que chaque équipier envisage peuvent être comprises comme autant de contraintes supplémentaires qu'il y a d'autres membres de l'équipe. Ainsi, l'équipe doit non seulement faire face aux contraintes imposées par le contexte du projet (*essentielles*) mais aussi à une deuxième catégorie de contraintes (*accidentelles*) provenant des préférences de chacun, des expériences vécues, de la perception individuelle des priorités enfin.

Ainsi, quelle que soit la configuration de l'équipe et la méthode de travail qu'elle adoptera, agile ou pas, celle-ci ne pourra faire l'économie d'un travail d'alignement, de création d'un langage et d'une expérience commune. Ce travail est long, complexe et frustrant, mais il n'en demeure pas moins indispensable si l'équipe souhaite créer de la valeur – la raison pour laquelle elle s'était constituée au départ.

Knocking at the Heaven's Door

Nous avons présenté différentes stratégies que les équipes de développement logiciel mettent en place dans l'espoir de mener les projets à bon port : documenter, regrouper, fusionner. Ces stratégies nous laissent penser que les membres des équipes envisagent leurs incompréhensions mutuelles comme un obstacle à contourner.

Nous souhaitons ici nous éloigner de cette conception de la situation et en proposer une autre.

Un client exprime à une équipe de développement un besoin qui sous-tend (disons) une dizaine de contraintes. À l'instant où il exprime ce besoin, il agite dans l'esprit collectif de cette équipe cent, mille contraintes dont il n'avait pas conscience – et les germes de frustrations à venir. L'équipe pourrait chercher à ignorer, masquer ou supprimer ces contraintes surnuméraires. Nous suggérons à l'inverse qu'elle les accepte pour créer l'espace dans lequel chacun se sent reconnu, et incité à contribuer à la production d'une solution satisfaisante pour l'utilisateur. Loin d'être un obstacle à contourner, les frictions dans l'équipe – tout au moins, celles liées à la compréhension du problème posé – constituent une porte qui, une fois ouverte, peut laisser place à des solutions génératrices de valeur pour l'utilisateur.

Les pratiques XP favorisent les opportunités de communication au sein de l'équipe. Elles n'évitent pas pour autant les frictions dans la résolution du système que constitue les différentes contraintes (techniques et fonctionnelles) que les membres de l'équipe perçoivent. Nous suggérons de ne pas chercher à *supprimer* ces frictions, nécessaires, préliminaires à un résultat significatif. De même, la pertinence des pratiques agiles – en particulier client sur site, binômage, métaphore... – ne doit pas être jugée à l'aune des “conflits” que celles-ci révèlent.

“**Client sur site**” permet aux développeurs de lever informellement et très rapidement les moindres incertitudes sur le comportement attendu de l'application en construction. Cela leur permet de rester concentrés sur les contraintes techniques sans avoir besoin de prendre en compte *a priori* l'ensemble des contraintes fonctionnelles.

Si l'on accepte avec les méthodes agiles que “le code est la documentation”, la pratique de “**binômage**” offre l'opportunité au rédacteur d'obtenir le *feedback* du lecteur (son binôme) au moment même où le “document” est rédigé. Le rétablissement d'un retour d'information rapide limite les risques d'incompréhension.

“**Métaphore**” incite l'ensemble de l'équipe à s'aligner sur une compréhension partagée de ce que l'application fait et la manière dont celle-ci le fait. Puisque le langage des spécialistes métier est relativement inaccessible aux développeurs et vice-versa, la métaphore agit comme un langage intermédiaire, à mi chemin, à partir duquel chacun peut tirer ses propres parallèles.

Envoi

Les méthodes agiles (et XP en particulier) emportent l'adhésion et suscitent la critique parce qu'elles renvoient à des valeurs qui sont propres à chacun, qu'elles les honorent ou les remettent en cause.
En sous-titrant le livre introduction d'XP “Embrace change”, Kent Beck plante les ferments du débat d'idées qui s'en est suivi depuis.

L'appréhension du système de contraintes que l' “autre camp” perçoit demande à ce que nous puissions envisager la construction logicielle en cours selon un *autre* point de vue, le leur. “Accepter le changement” n'est pas forcément une *valeur* qu'on doit chérir ou dont on doit se méfier, c'est peut-être simplement une *pratique* nécessaire dans la construction d'une vision commune entre ceux qui définissent le “quoi” et ceux qui définissent le “comment” d'un projet.

Références

Alistair Cockburn, Agile Software Development: The Cooperative Game, 2e éd. 2006, éd. Addison-Wesley Professional

Eric Evans, Domain Driven Design: Tackling Complexity in the Heart of Software, 2004, éd. Addison-Wesley

Mary et Tom Poppendieck, Implementing Lean Software Development: From Concept to Cash, 2007, éd. Addison-Wesley

Gerald Weinberg, Quality Software Management vol. 1: Systems Thinking, 1992, éd. Dorset House