Analogies entre l'équipe et son logiciel *(seconde édition)*

Emmanuel CHENU
emmanuel.chenu@gmail.com
http://emmanuelchenu.blogspot.com/

Cet article est une réflexion sur les similitudes qui existent entre une équipe de développement logiciel et son produit. Les bonnes pratiques de génie logiciel s'appliqueraient-elles également à l'équipe?

L'organisation de l'équipe et de son logiciel se ressemblent

En lisant le « *Mythe du Mois Homme* » de Frédéric Brooks[1], j'ai découvert la Loi de Conway[2]. Elle prédit que les organisations qui conçoivent des systèmes les réalisent à l'image de leurs propres structures. L'homme créé à son image.

J'ai eu l'occasion constater l'application de cette loi lors d'un grand projet multi-site. Les deux équipes créant le même logiciel se méfiaient l'une de l'autre. D'ailleurs, il se trouve que notre partie de l'application était très méfiante du reste du logiciel. En effet, une façade était le seul moyen de s'interfacer avec notre composant. Nous avions appliqué une programmation par contrat avec levée d'assertion dès qu'une pré-condition n'était pas respectée par l'utilisateur. Il s'agit d'une programmation sans confiance et avec sanction immédiate.

Il me semble que prendre conscience de la Loi de Conway aide à prendre du recul sur l'organisation du logiciel en cours de développement et ainsi que sur l'organisation de l'équipe qui créé le logiciel. Est-ce que certaines bonnes pratiques de génie logiciel seraient également applicables à l'équipe de développement?

La première organisation du logiciel n'est pas la bonne. Qu'en est-il pour l'équipe?

Aussi, Conway remarque qu'une première conception est presque immanquablement pas la bonne. La première conception doit changer et s'adapter pour converger vers une meilleure conception.
En se référant à la Loi de Conway, quelle conclusion pouvons-nous faire sur l'organisation qui a créé cette première conception inadéquate? Si elle est comparable à son produit, elle est elle aussi sûrement inadéquate! La première organisation n'est pas la bonne et elle doit changer et s'adapter pour converger vers une meilleure organisation. Une meilleure organisation qui concevra un meilleur produit.

Classiquement, là où je travaille, nos équipes projet ont une organisation comptable. Un budget est alloué à chaque spécialité *(études, développement, moyens d'essai, moyens informatiques ...)*. L'équipe est donc organisée et pilotée par activité. L'équipe est elle-même une cascade! Petit à petit nos organisations convergent vers de vrais équipes pluridisciplinaires avec moins de chefs et moins de frontières.

Le remaniement du logiciel. Et de l'équipe?

Désormais, nous avons bien pris conscience de la nécessité de remanier en continue le logiciel.
La Loi de Conway nous amène à prendre conscience qu'il faut également remanier en continue l'organisation qui créé un logiciel. La démarche Scrum a formalisé ce concept par la tenue des rétrospectives d'itération. Régulièrement, l'équipe réfléchit à la manière de se réorganiser pour travailler mieux. Elle expérimente ces améliorations lors de la prochaine itération.

La construction incrémentale du logiciel. Et de l'équipe?

Désormais la montée en charge des capacités d'un système se fait couramment de manière itérative et incrémentale.
Notre équipe de développement applique cette même démarche pour elle même. Pour absorber la charge de travail, l'équipe croit par petit incrément au fur et à mesure des itérations. Nous constatons

[1] Voir référence bibliographique en fin d'article.
[2] Conway's Law: Organizations which design systems are constrained to produce systems which are copies of the communication structures of these organizations.

empiriquement un rythme maximal de croissance de un développeur par mois pour une équipe de moins de neuf personnes. Ainsi, si besoin, l'équipe monte en capacité de manière itérative et incrémentale avec une vélocité maximale de 1 développeur par mois et par groupe de neuf développeurs.

L'organisation par faible couplage et forte cohésion

Il est admis qu'un logiciel se découpe en modules selon des critères de faible couplage et de forte cohésion. Cette pratique s'applique aussi pour la modularisation d'une grande équipe en sous-équipes.
Si la dimension de l'équipe vient à dépasser la taille idéale de 7 +/- 2 développeurs, il s'agit alors de la découper en deux équipes de 7 +/- 2 membres selon des critères de forte cohésion fonctionnelle et de faible couplage. Il faut éviter d'organiser l'équipe selon des critères de spécialité ou d'architecture.
Le faible couplage entre les équipes n'a pas pour objectif de réduire la communication, mais plutôt de réduire les zones d'intervention commune. Les équipes devront donc être réorganisées au fur et à mesure que l'architecture du produit émerge.

Le pilotage par les tests du logiciel. Et de l'équipe?

Avec l'eXtreme-Programming en particulier, le test du logiciel est reconnu comme primordial. Le test devient même un outil de pilotage du projet. Existe t-il une notion de pilotage de l'équipe par les tests?
Dans notre équipe de développement, nous mesurons la vélocité de l'équipe, c'est à dire sa productivité. Il s'agit d'une métrique factuelle. On peut assimiler la mesure et le suivi de la vélocité à un test de l'efficacité de l'équipe. Lorsque le test indique une « défaillance », l'équipe cherche à se réorganiser pour améliorer son efficacité. La gestion de l'équipe est alors pilotée par des tests de l'équipe.

Intégration continue du logiciel. Et de l'équipe?

La pratique de l'intégration continue permet de minimiser les désynchronisations entre les développements sur un même logiciel. Elle permet aussi d'incrémenter la capacité du logiciel en flux continu ce qui évite la dangereuse accumulation de travaux inachevés.
L'intégration continue est-elle applicable à l'équipe de développement?
Scrum propose la pratique du « daily stand-up meeting » ou « scrum meeting ». Chaque équipier rend public l'état ses tâches et est tenu au courant des travaux des autres. L'objectif de cette courte réunion quotidienne est donc de conserver une équipe synchronisée. De plus, rendre public les tâches permet de concentrer l'effort sur la terminaison des tâches en cours pour éviter la dispersion et l'accumulation de travaux inachevés. Ainsi, on peut assimiler le « daily stand-up meeting » à l'intégration continue de l'équipe de développement.

Contrôle et délégation

En conception logicielle, il faut se méfier de la prolifération de classes « controller » qui pilotent d'autres classes. En effet, la technologie orientée-objet pousse à une plus grande responsabilisation de chaque module de l'architecture du logiciel. Chaque classe est une entité autonome, avec ses responsabilités, exposant une partie publique minimaliste et cachant dans sa partie privée les mécanismes de son organisation intime. Il est de bon ton de concevoir en responsabilisant et en délégant.
Ces pratiques de génie logiciel sont-elles également applicables aux équipes de développement?
Des démarches telles que le Lean et le développement Agile mettent en effet l'accent sur l'autonomie et la responsabilisation des équipes de développement. D'ailleurs, le management situationnel[3] recommande une attitude de management délégatif pour des équipes efficaces et motivées.

Défauts et conflits

Les démarches agiles partagent la propriété de détecter tôt les premiers défauts introduits dans le produit. De même, nous avons remarqué que les équipes agiles lèvent les conflits plus tôt dans le projet. Il peut s'agir de problèmes de communication, de collaboration, d'organisation, de travail d'équipe, de valeurs partagées ou non au sein de ses membres.
Les défauts sont détectés tôt car l'équipe produit rapidement un logiciel opérationnel qui fournit un retour d'information. Ceci est aussi la cause de la levée rapide des conflits. Livrer au plus tôt un produit oblige à exercer au plus tôt toutes les formes de collaboration, de communication et de travail d'équipe nécessaires à la création de valeur pour le client.

[3] Voir http://en.wikipedia.org/wiki/Situational_leadership_theory

Ainsi, le conflit est à l'équipe ce que le défaut est à son produit. Si les défauts sont évités de manière préventive par une pratique continue des tests alors les conflits sont désamorcés par une pratique régulière des rétrospectives d'itération.
De plus, si un conflit est à l'équipe ce qu'un défaut est à son produit, alors plus il est identifié tôt, plus son impact sera limité. Ce constat milite encore en faveur des démarches agiles qui provoquent ce retour d'information au plus tôt.

Équipe = Produit

Lors de sa conférénce sur Crystal à l'Agile Tour 2008 à Valence, Géry Derbier a présenté l'équation « Équipe = Produit ». Si nous admettons que le développement logiciel est un jeu coopératif d'invention et de communication, alors nous sommes amenés à accepter que produit ne peut-être de meilleure qualité son équipe. Une équipe défectueuse élabora un produit défectueux. Le produit aura au mieux les vertus que son équipe est capable d'y insérer. Cette dualité peut être utile pour identifier et analyser des dysfonctionnements. Pour détecter ceux du produit, nous pouvons observer ceux de l'équipe et inversement. Ce constat nous ramène à Loi de Conway. La boucle est bouclée.

Il existe sûrement d'autres bonnes pratiques du génie logiciel qui s'appliquent de manière pertinente à l'équipe de développement. De même, on doit aussi pouvoir appliquer des règles d'organisation, de management et de travail d'équipe au développement de logiciels. Ces petites prises de conscience feront l'objet de nouvelles éditions de ce petit article – qui se construit de manière itérative et incrémentale ;o)

BIBLIOGRAPHIE

The Mythical Man-Month, Frederick P. Brooks, Jr. ISBN 0201835959