Binômage : que multiplie-t-on par deux ?

Pour tenter de comprendre en quoi cette pratique est la haute couture des pratiques Extreme Programming.

Ce document a été rédigé en collaboration
avec Nicolas Charpentier (Charpi.net)

Vous pouvez obtenir ce document à l'adresse suivante :
http://www.chiroa.com/sas/xp-binomage.doc.pdf
Révision 1.0
26 octobre 2007
18 pages

Jacques COUVREUR ● jacques@2ia.net (((Chiroa.com)))

Table des matières

Table des matières

Index des illustrations

Présentation

Le binômage est doublement l'apanage des privilégiés.

Tout d'abord, par définition, c'est l'une des pratiques que l'on ne peut pratiquer (et donc améliorer) seul. On peut faire des tests unitaires seul ou s'imposer une conception simple, mais il est difficile de s'entraîner seul au binômage sans être schizophrène ou le devenir.

De plus le binômage est la partie (la plus) visible de l'iceberg XP. En ce sens il est difficile de le mettre en pratique sans l'aval de sa direction. Il est aisé de faire du *refactoring*, le soir en « sous-marin » sur ses heures sup' ou d'installer un serveur d'intégration continue sur le PC familial, mais il est difficile de rester durablement à deux devant une machine sans que cela ne soulève des questions, voire des réprimandes.

Mais le binômage est doublement sensible.

Comme souvent c'est l'aspect financier qui fait du binômage l'une des pratiques les plus polémiques et controversée par ses détracteurs. Qui n'a jamais entendu (ou s'est même demandé si) « Binômer c'est multiplier les coûts par 2 » ?

Cela occulte trop souvent l'autre challenge. Pour tenter l'aventure du « *Pair Programming* » de façon efficace et durable, on doit considérer le binôme comme une paire non plus de « ressources », mais bien de « personnes », avec la dimension relationnelle que cela implique.

En résumé « quelle différence y a-t-il entre 2 personnes devant un PC et du binômage ? »

Le binômage ça n'est pas...

Le binômage est donc une des pratiques agiles les plus controversées. Parmi ses détracteurs, certains ont une expérience plus ou moins approfondie du binômage. Parmi ces derniers encore, cette expérience, plus ou moins bien vécue, a amené cette image négative du binômage, ou du moins non primordial. Si l'on essaye de comprendre leurs objections, on s'aperçoit qu'ils étaient confrontés, bien souvent, à des équipes qui ne pratiquaient pas un « binômage efficace ».

Le binômage est une discipline complexe qui ne peut être maîtrisée immédiatement. Chaque équipe va développer sa propre pratique du binômage, mais, si elle n'est pas vigilante, elle va développer des « anti-patterns ».

Or, ces derniers participent pour beaucoup à la « mauvaise » réputation du binômage.

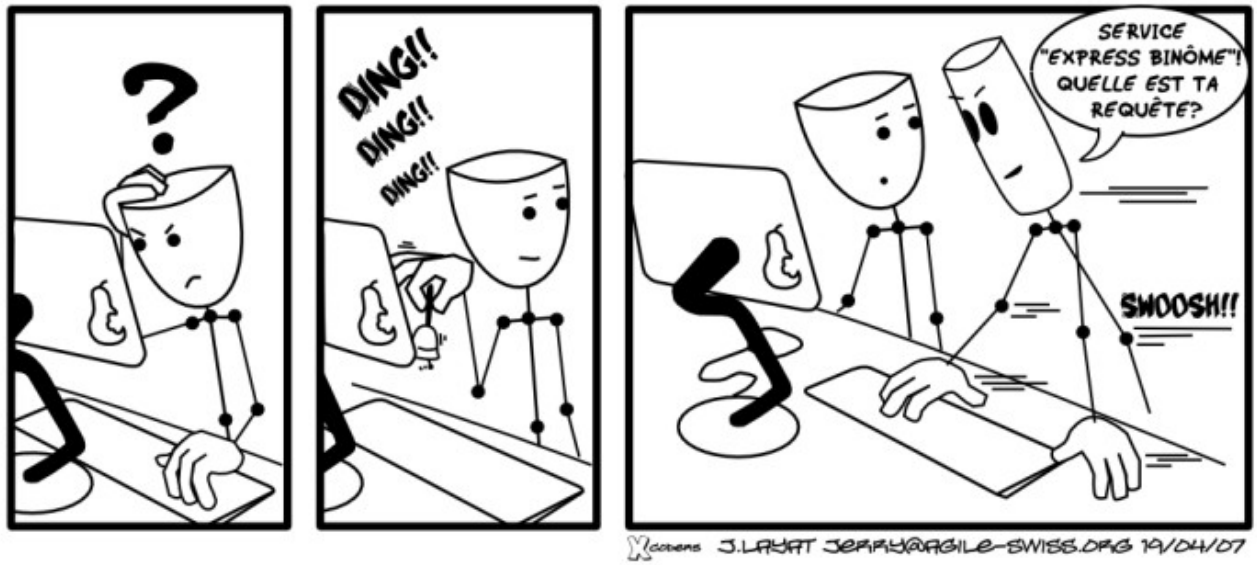


Illustration 1: Le binômage ca n'est pas du support express

La première idée reçue sur le binômage est qu'il se réduit à du « **support ponctuel** ». Souvent, alors que vous parlez binômage, votre interlocuteur vous lance « moi-même, il m'arrive de binômer. Hier encore, nous avons passé 3 heures à deux sur la machine pour résoudre un problème ardu ». Cela est plus exactement « du debug à deux », souvent avec le même « expert gourou ».

Mais le binômage, ça n'est pas que ça.

Un autre cliché, mainte fois énoncé : le binômage c'est « **un qui travaille pendant que l'autre dort** ». Si l'autre dort effectivement (ou n'est simplement pas concentré), ce n'est pas du binômage, mais du « monômage » classique. Mais un développeur qui

dort, est de toute façon pas une bonne chose, quelque soit la façon de développer !

Par contre, si l'un utilise le clavier pendant que l'autre surveille et réfléchit à ce que fait le premier, cela devient du développement avec relecture de code en temps réel.

Mais le binômage ça n'est pas que ça.

Dans la même veine, « **deux personnes qui parlent ensemble, c'est une perte de temps** ». Si ces personnes bavardent effectivement plusieurs heures par jour, sur la défaite de la France en ½ finale de la coupe du monde de Rugby, c'est effectivement une perte de temps… Et cela, encore une fois, quelque soit la méthode de développement.

Par contre, si ces personnes parlent de conception, pour une tâche donnée, ce n'est plus une perte de temps, mais bien de l'analyse et conception en amont de la réalisation technique, ce qu'il est difficile de faire seul. Le binômage permet de favoriser cet échange en « rapprochant » les personnes.

Mais le binômage ça n'est pas que ça.

Aller, une dernière. Ceux qui ont une petite expérience du binômage et des problématiques liées à sa mise en oeuvre, s'exclament parfois « **je ne vais pas mettre mon expert avec mon débutant, cela va ralentir l'expert** ». Cela va aussi lui permettre de se remettre en question suite aux questions du débutant, améliorer sa pédagogie. Et puis surtout, le débutant va monter en compétence, tant technique que fonctionnelle. Cela s'appelle de la formation continue...

Mais le binômage ça n'est pas que ça.

On constate donc que « il n'y a pas de fumée sans feu » : ces *anti-patterns* et autres idées reçues sont bien issus de dérives d'aspects qui se veulent la force du binômage. Il semblerait donc que le binômage soit un amalgame complexe de composantes diverses...

Les moments de la vie

Pour appréhender toute la dimension du binômage et commencer à apercevoir les intérêts, mais aussi les problématiques, d'un binômage efficace, il est important de parcourir les moments de la vie d'un binôme. Pour cela, plaçons-nous dans une équipe *Extreme Programming*.

Le planning game

En tout début de projet, et à chaque début d'itération, l'équipe se retrouve pour une « réunion » qui dure de ½ journée à plus d'un jour selon taille de l'équipe, la durée de l'itération et la complexité de ses scénarios : c'est le *planning game*.

Une fois les scénarios exposés par le client, les développeurs les découpent en tâches qu'ils estiment. Pour cela, les membres de l'équipe posent les questions nécessaires au client pour bien comprendre l'aspect fonctionnel. Ensuite, ils confrontent leur connaissance de l'application et leur vision de la réalisation technique, afin de s'accorder sur une estimation, en jour-binôme. Cette estimation est considérée comme le nombre de jours qu'il faudrait à un binôme pour réaliser cette tâche en s'y mettant le lendemain matin, binôme constitué d'un débutant et de l'expert approprié.

Ainsi, n'importe quel « ½ binôme » est susceptible de travailler sur n'importe quelle tâche et fonctionnalité.

On comprend ici, que, dès le départ du projet ou de l'itération, le futur binôme a en main les clefs de sa destinée.

Le stand-up meeting

Au delà de la planification de l'itération durant le *planning game*, l'équipe se réunit quotidiennement, pendant 10 à 20 minutes maximum (selon la taille de l'équipe) : c'est le *stand-up meeting*.

L'objectif de cette réunion, est, pour chacun des binômes, de :

1. Résumer la journée passée
2. Planifier la journée à venir
3. Identifier ce qui empêche d'avancer
4. Voir ce qui pourrait empêcher la livraison de fin d'itération.

Cela permet de réaliser au mieux l'objectif ultime de cette réunion : former les binômes de la manière la plus adéquate pour la journée à venir.

Une façon efficace d'y parvenir est de mettre tout le monde debout (d'où le nom !), en

cercle. Chacun aborde alors, tour à tour, les 4 items exposés ci-dessus. Puis une fois le tour de table terminé, on forme les binômes pour la journée avant de se séparer.

La difficulté de cette pratique, est de trouver le compromis entre passer suffisamment de temps à discuter pour identifier les bugs latents, et ne pas se perdre dans des détails techniques ou d'interminables discussions, afin de ne pas s'éterniser : 8 développeurs qui discutent pendant 30 minutes, c'est 4h par jour passées à ne pas développer... Il vaut mieux que cela soit pour une bonne raison !

Le *stand-up meeting* est vraiment le trait d'union entre deux journées de travail. Il peut donc se faire le matin, le soir ou le midi, tout dépend de ce que l'on considère comme une journée de travail. C'est un peu comme les années civiles et les « saisons sportives » à cheval sur deux années civiles.

C'est également LE lieu et L'instant privilégié pour l'échange, la communication. On peut poser telle question à l'équipe, rappeler telle information ou telle astuce technique concernant une tâche.

Mais on ne s'éternisera pas sur une problématique de conception ou à vouloir résoudre verbalement un bug : si la conversation commence à durer, on coupe court, quitte à former un binôme « mieux » adapté à la résolution de ce bug, ou à improviser une réunion de conception après le *stand-up*.

Si des membres de l'équipe ont une compétence permettant de faciliter le travail d'un binôme, ils le signalent, afin que, si besoin, dans la journée, le binôme puisse venir en profiter.

Il faut bien voir le *stand-up* comme une opportunité quotidienne de prendre de la hauteur par rapport à sa tâche, par rapport au fonctionnement de son binôme mais aussi de l'équipe.

Tout cela ne doit pas dépasser les 20 minutes grand maximum, au bout desquelles l'équipe se sépare en de nouveaux binômes, adaptés, sachant précisément ce qu'ils doivent faire et sur qui ils peuvent compter pour y parvenir : si ça, ça n'est pas déjà un défi !

La réalisation de la tâche

Une fois le binôme formé, il s'assoit devant une des machines de développement disponibles, et en maximum 10 minutes (le temps de prendre un café), il a récupéré la dernière version des sources de l'application et est prêt à travailler. On dit 10 minutes si on prend le temps de jouer tous les tests... Un environnement efficace et adapté s'impose donc... Mais nous y reviendrons...

Dans un monde XP idéal, le binôme commence par prendre le test fonctionnel correspondant à la tâche qu'il doit réaliser. Par expérience, une autre technique

permet de « couper la poire en deux » : le binôme peut « amener » le client sur son poste pour écrire le test fonctionnel avec lui. Il faut bien comprendre que le test fonctionnel est de la responsabilité du client, mais qu'il peut être réalisé par l'équipe.

Donc, le test (fonctionnel) est écrit avant la réalisation technique... Et oui, nous travaillons en *Test Driven Development* (TDD). Cela vaut tant pour les tests fonctionnels avec la tâche technique, que le test unitaire avec la classe ou la méthode/fonction. Pour le test unitaire, on peut même aller plus loin : le développement « ping pong ». Le pilote écrit un test unitaire de méthode ardu puis donne le clavier au copilote qui va alors tout faire pour faire passer le test au mieux avant d'écrire le test unitaire suivant... Et ainsi de suite... Etant donné qu'il existe un ouvrage de référence complet sur le TDD, nous n'allons pas avoir la prétention de traiter ce sujet en un paragraphe. Nous voulions juste attirer votre attention sur l'intérêt du développement « ping pong » quant à la synergie qu'il peut induire.

Au delà du développement proprement dit, si le binôme n'est pas sûr d'un choix qu'il vient de faire, ou que cela fait maintenant plus de 30 minutes qu'il débat sans parvenir à un consensus, il peut improviser une réunion en conviant un ou plusieurs autres membres de l'équipe. La difficulté ici est d'identifier les personnes qui ont le plus de valeur ajoutée pour la problématique abordée. C'est là qu'un *stand-up meeting* bien mené peut porter ses fruits : il aura permis au binôme d'identifier les personnes adéquates.

Et si la problématique n'est pas d'ordre technique mais fonctionnel, le client sur site, disponible, aura pour mission d'y répondre.

Quelle journée !

La (dé)formation ou rotation

Bien souvent, les nouveaux binômes sont donc formés au moment du *stand-up meeting*. Mais il arrive que l'on doive former un nouveau binôme dans la journée, pour des raisons diverses et variées, mais le plus souvent parce que la tâche est terminée.

Dans ce cas, avant de se séparer, le binôme s'assure qu'il a bien remonté l'ensemble de ses sources sous le gestionnaire de version, et donc forcément cela signifie que tests unitaires et tests fonctionnels sont tous « en succès ». Ce « moment de la vie » a donc un effet important : il a pour conséquence le déroulement d'un cycle d'intégration complet, ce que certains projets sont incapables de garantir en amont d'une phase de recette finale. De plus, le fait de scinder ce binôme assure une triple diffusion de la connaissance : chacun des deux membres du binôme va en effet pouvoir propager « la bonne parole » tout comme le code qui vient d'être remonté sous le gestionnaire de version.

Il est donc important de surveiller à ce qu'un binôme ne persiste pas trop longtemps. On aborde là un sujet polémique, même au sein de la communauté XP : la durée de vie d'un binôme. Certains pensent que l'on doit laisser un binôme terminer sa tâche, sachant qu'une tâche ne doit pas dépasser quatre jours. D'autres assimilent les binômes aux éphémères, ces insectes dont la durée de vie n'excède pas la journée.

Il est vrai, que lorsqu'un binôme termine sa tâche en milieu de journée, on hésite toujours à « casser » un autre binôme afin de permuter les développeurs. Mais il faut comprendre que cela porte ses fruits sur le moyen terme.

Dans tous les cas, si le binôme n'a pas la compétence (technique ou fonctionnelle) pour traiter la nouvelle tâche, la question ne se pose pas : la priorité est toujours de former un binôme adéquat pour une tâche donnée.

Il apparaît donc pertinent de ne pas sous-estimer l'importance de la rotation.

Cycle de vie

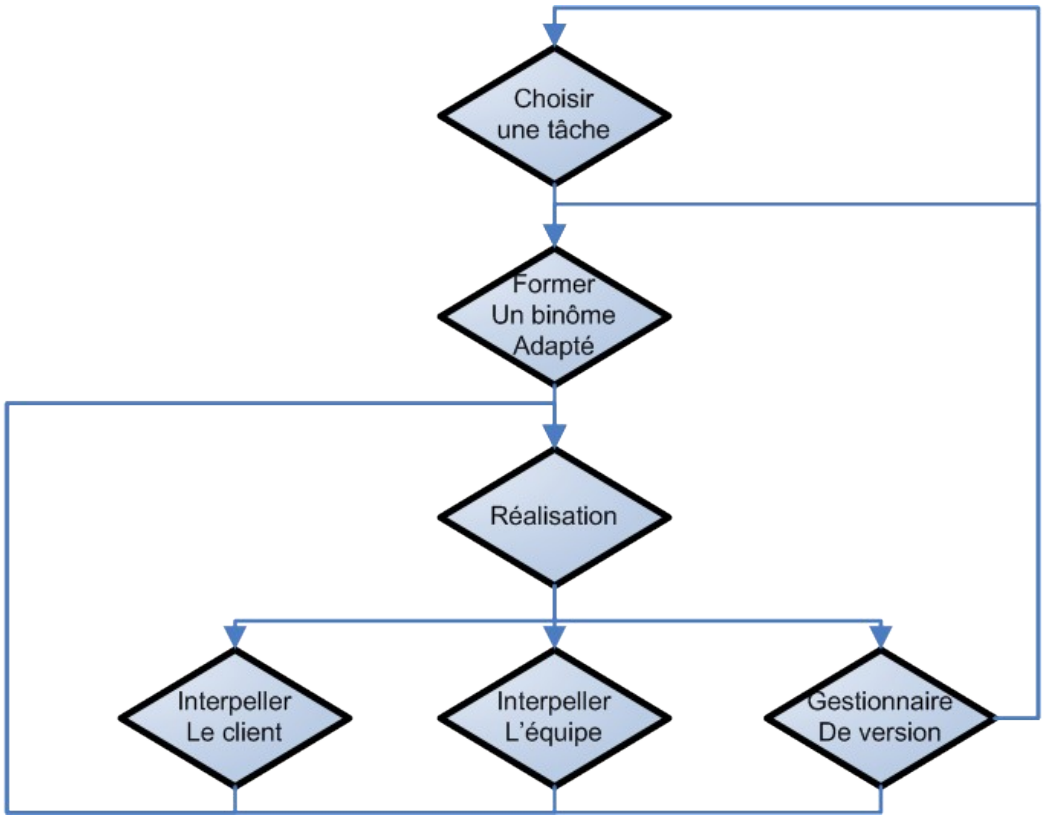


Illustration 2: Cycle de vie d'un binôme

Intérêts implicites et coût véritable

Les avantages qu'apporte le binômage sur le moyen terme sont souvent imperceptibles à court terme, parfois difficilement quantifiables, mais assurément nombreux.

La « **cohésion de groupe** ». A mesure que le projet avance, que les tâches se succèdent, que les binômes se forment, se déforment et se reforment, on apprend à se connaître. Le temps que se passent les compilations, les exécutions des tests unitaires, les lancements de l'application, ou encore le temps d'une pause café, on fait chaque d'avantage connaissance avec son « ombre ».

La « **communication verbale** » doit être et est favorisée par rapport à la documentation et les outils. Le fait de travailler durablement à deux personnes proches, une discussion permanente s'établit avec le binôme, mais aussi entre binômes grâce à la « *war room* ». Cette communication verbale est entretenue via différents moyens tels que la rotation des binômes, le *stand-up meeting*, le *planning game*, la *war room*, des développeurs curieux et courageux...

Le fait d'être à deux sur une tâche technique a cet autre avantage de garantir une « **meilleure réutilisation** ». Il y a deux fois plus de chances qu'un membre du binôme sache comment utiliser tel framework, ou même que telle librairie est déjà utilisée pour tel besoin. Lorsqu'un problème survient, qu'un bug apparaît, qu'un piège guette, il y a deux fois plus de chances que la bonne solution, identifiée précédemment, soit redéployée.

Cela supprime « **l'expert boite noire** », celui qui est le seul à connaître telle partie de l'application. En effet, le fait de favoriser la communication assure un transfert tant technique que fonctionnel. Cette « transparence » est renforcée par l'association d'autres pratiques XP, telles que la propriété collective du code, les tests, le client sur site...

Les développeurs vont également avoir une « **meilleure vue d'ensemble de l'application** » grâce au « **caractère fortement didactique** » du binômage. Lorsque l'on binôme sur une nouvelle tâche, on apprend d'avantage et plus vite. Le fait de tourner fréquemment les binômes permet de diversifier et d'accroître le nombre de « thèmes de formation ». Attention à garder l'équilibre sur la durée des tâches pour que le développeur ait le temps d'assimiler : le binôme a pour but d'achever au mieux la tâche ! Enfin, le « client sur site » est une source permanente

de connaissance fonctionnelle.

Et en l'absence de référents, technique et fonctionnel, l'équipe acquiert une « **résistance au turn-over** ». La diffusion, la répartition, de la connaissance à l'ensemble de l'équipe rend moins handicapant, voire transparent, le départ de tout membre de l'équipe.

Une autre conséquence directe de meilleure connaissance, fort appréciée des décideurs, est que l'équipe produit de « **meilleures estimations** ». En effet, chaque membre se forge une connaissance d'une grande partie de l'application. Le fait de pouvoir les confronter au cours du *planning game*, permet de produire des estimations au plus juste, directement déduites de l'expérience de chacun.

S'ajoute à cela l'unité de l'estimation, le « jour-binôme ». Il s'agit du temps que mettrait le débutant, aidé de l'expert approprié de l'équipe, pour réaliser la tâche : en temps que moyenne, l'estimation a plus de chance d'être juste.

Le binômage crée une certaine inertie, qui assure une « **productivité soutenue** » au quotidien. En effet, le fait d'être à deux, limite les baisses de régime : on hésite toujours à ralentir, voire faire autre chose, quand on est à deux sur un développement. Dans le cas contraire, le fait que l'autre continue assure une production lissée. Cela reste vrai sur une plus grande échelle de temps : que se soit sur les congés ou même les départs (*turn-over*), nous avons vu que la diffusion de la connaissance garantissait une certaine résistance.

Un autre facteur qui pousse les binômes à tenir le rythme, qui les motive à tenir les délais, est leur implication liée au fait que se sont les binômes eux-mêmes qui ont estimé les tâches durant le *planning game*.

Le fait de condenser ces pratiques dans le temps, l'espace et les personnes permet de « **réduire les temps de traitement** ». Moins de temps entre l'identification d'un bug et sa résolution, entre un l'apparition d'un problème et sa nouvelle réapparition, entre la réalisation d'une fonctionnalité et la réalisation de son évolution, en accélère le traitement.

Pour maintenir dans le temps et optimiser tous ces aspects bénéfiques, il est important « **d'entretenir une certaine dynamique** » dans le binômage. Cela tant au sein d'un binôme (via le développement « ping pong » par exemple) qu'entre les binômes (via des rotations fréquentes et pertinentes, sur les tâches, les fonctionnalités, les personnes). En gardant à l'esprit que le binômage est une pratique éprouvante, on fera donc attention à conserver un « **rythme durable** » (pas trop souvent d' « heures supplémentaires »).

Tout cela converge vers un but ultime : une « **meilleure qualité** ». L'émulation positive du développement « ping pong », le remaniement continu allié à la propriété collective par des binômes qui tournent fréquemment, le fait qu'on hésite moins à s'attaquer à un problème lorsque l'on est deux, et qu'alors le problème est plus vite résolu, tout cela conduit à un code de meilleure qualité.

Cet aspect est facilement mesurable, de façon factuelle, par exemple via le nombre de defects en retour de chaque livraison.

Conséquence directe pour l'équipe : meilleure qualité entraîne moins de debug, moins de maintenance, donc « **moins de dépenses** ».

Conséquence directe pour le client : meilleure qualité entraîne moins de bug, donc d'avantage de satisfaction, donc un « **meilleur capital confiance** ».

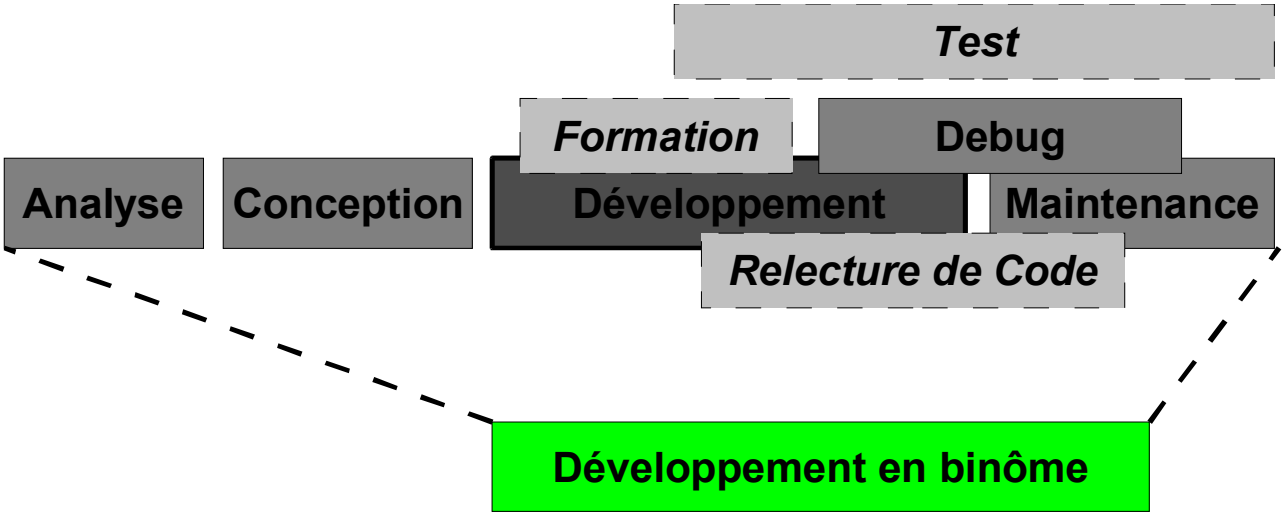


Illustration 3: Intégration des disciplines de développement

Mise en place (outillage & profils)

Maintenant que vous êtes convaincu du bien fondé du binômage, il vous faut le mettre en place. Pour cela il est important de bien s'outiller, tant sur l'aspect matériel que « humain ».

L'environnement de travail

Pour cela, il vous faudra un environnement :

- Adapté,
- Efficace
- Homogène.

En cherchant un environnement adapté, il y a de fortes chances pour que vous convergiez vers une « trousse à outils » minimale, à savoir « une boite à tâches » qui permet de saisir : les tâches à réaliser, leurs estimations (« temps binôme », risque...) et le « temps binôme » de réalisation. Du coup, vous aurez également un outil de tracking intégré !

Vous aurez également mis en place une intégration continue... donc des tests unitaires et fonctionnels automatisés. Cette intégration continue est d'autant plus importante que le travail en binôme s'envisage difficilement sans une propriété collective du code, et donc l'intervention plus ou moins fréquente d'un binôme sur une portion de code qu'il découvre plus ou moins et ne maîtrise pas. Des tests unitaires et une intégration continue permettront de lever au plus vite tout oubli ou erreur. On touche ici à l'efficacité.

Autre axe d'efficacité, l'IDE. Hélas bien souvent le choix de l'IDE est conduit par des contraintes économiques, voire politiques, au sein de l'entreprise. Et si ce n'est pas le cas, il est induit par des questions pas forcément des plus pertinentes : « Est-ce que l'on peut faire des diagrammes UML et de la génération de code avec ? ». « Pouvons-nous avoir du support dessus ? ». Il y a des questions bien plus fondamentales qui doivent guider le choix d'un « bon » IDE pour le développement en binôme : « Sachant qu'il intègre forcément les *refactorings* en natif, ces *refactoring* sont-ils performants ? » ou encore « Est-il facilement paramétrable afin d'assurer une configuration identique sur tous les postes ? ».

A ce niveau on aborde la notion d'homogénéité. Étant donné que les binômes tournent fréquemment, un développeur est amené à changer fréquemment de poste de développement. Il est donc important de garantir que tous les postes ont une configuration identique. Cela passe par des pratiques techniques telles que la configuration de l'IDE et des pratiques d'ingénierie telles que les conventions de code.

Et pour ne pas nuire aux rotations fréquentes, l'environnement de développement dans son ensemble devra être performant. Pour commencer une tâche, un binôme qui arrive sur son poste, se contentera de faire un « update » sur le gestionnaire de version, lancer une construction du projet, s'assurer que tous les tests unitaires sont ok, ouvrir l'IDE avant, pendant ou après avoir lancé l'application en local : « et voilà », il peut travailler sur le poste de développement !

Toujours dans l'idée de faciliter le travail d'un binôme, il faut que l'environnement « géographique » soit adapté, on entend par là un bureau « au sens large » qui permette le travail à deux par poste et l'échange verbal. Je veux bien évidemment parler de la « *war room* ».

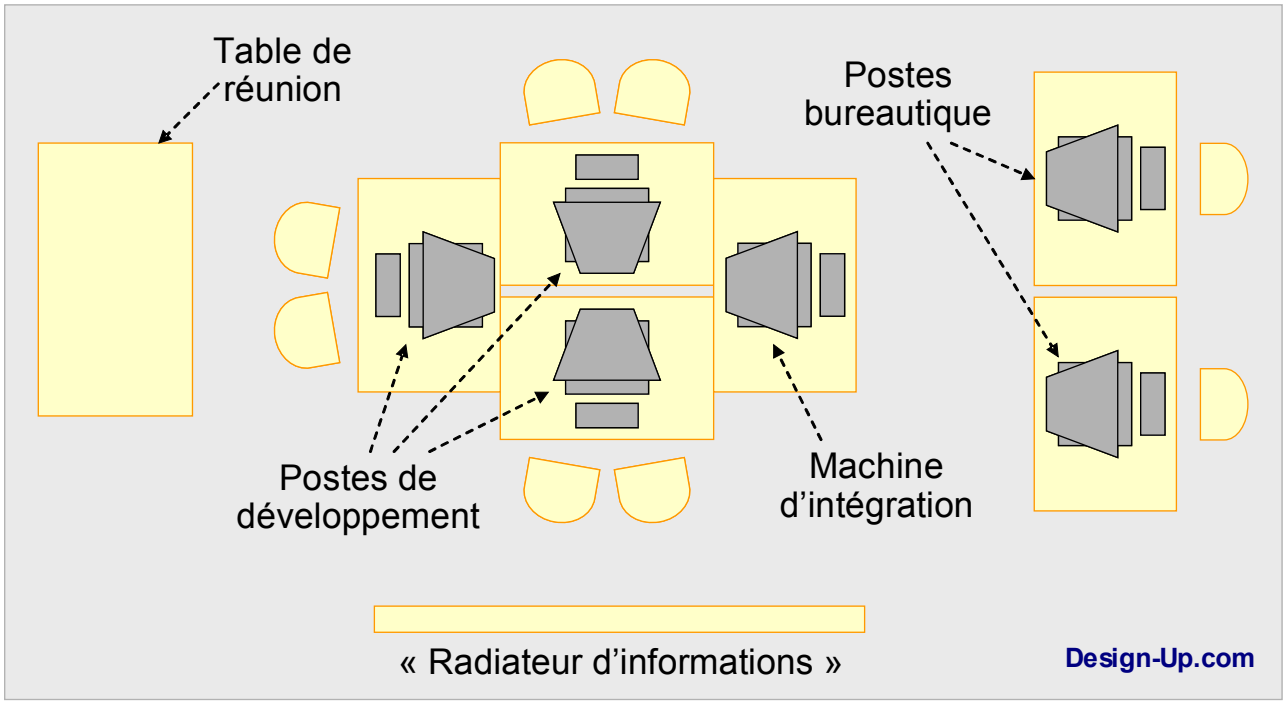


Illustration 4: La war room

Un bon profil

Une fois formée, il faut remplir la « *war* room » au mieux… Et nous ne parlons pas des meubles ! Trouver « un bon » profil est assurément le point le plus délicat quand on tente de mettre en place, ou d'améliorer, une équipe amenée à travailler en binômes.

Tout d'abord, il faut avoir conscience que le binômage est contraignant.

Le binômage est avant tout une pratique éprouvante : pendant huit heures de travail continu, les deux membres du binôme, côte à côte, sont tour à tour codeur et relecteur, épié et « épieur », pilote et copilote.

C'est également une pratique conflictuelle, dans le sens où elle remet en question les rôles de référent technique et fonctionnel, d'architecte logiciel, voire de chef de projet, ces rôles étant collectivement distribués.

Ainsi, le profil du « bon ½ binôme » présentera des valeurs et capacités essentiellement relationnelles et humaines plus que techniques :

- Ouverture d'esprit et remise en question
- Courage (de se mettre à nu)
- Communication active (pas de rétention d'information)
- Respect de l'autre et de son travail
- Capacité à tourner (tâches, fonctionnalités, personnes...)
- A se rendre non indispensable
- A travailler à deux
- A partager la gloire... Et les erreurs

Il apparaît en effet, par expérience, que « la technique, ça s'apprend » et qu'il est « plus facile de former un débutant (à l'agilité) que de déformer un gourou ».

Lors d'un entretien d'embauche d'un membre de l'équipe, il faudra donc rechercher ces « originalités » en gardant à l'esprit que l'on cherche à composer une équipe complémentaire. Des « ½ binômes » complémentaires sont donc au combien préférables à plusieurs développeurs géniaux. Pour exprimer cela autrement, prenons la métaphore du costume. Une magnifique chemise peut jurer et être dépareillée avec un costume tout aussi magnifique de son côté. Le défi est donc de parvenir à composer un ensemble coordonné avec un costume, une chemise, des chaussettes, une ceinture, des chaussures, une cravate… qui sont autant d'occasions de finir habillé comme un clown !

Conclusion

Le binômage est un révélateur intransigeant sur de nombreux aspects, que ce soit le comportement des membres de l'équipe, l'expérience et les faiblesses de chacun, les erreurs dans le code, les failles de conception de l'application, les faiblesses dans les processus... La moindre « anomalie » remontera de suite à la surface... Et tant mieux !

Nous avons vu que le binômage peut sembler au premier abord et à court terme plus onéreux si on ne considère par exemple que le nombre de lignes de code produites par personne. Mais dès lors qu'on se projette dans la durée et que l'on énumère l'ensemble des aspects bénéfiques qui en découlent en se basant sur des mesures comme le nombre de bugs identifiés après chaque livraison ou la « véracité » des estimations, le binômage apparaît alors comme un investissement éclairé qu'il est difficile de remettre en question.

Sans compter un autre facteur, aussi difficilement quantifiable qu'important, la satisfaction... du client, donc du chef de projet, donc de l'équipe.

Cerise sur la gâteau, le binômage prend tout son sens, voire ne peut s'envisager, sans coexister avec d'autres pratiques agiles : un rythme soutenable, la propriété collective du code, les conventions de code, les tests (unitaires et fonctionnels), le TDD (voire le développement « ping pong »), le remaniement continu, le *stand-Up meeting*, le client sur site, le *planning game*.

Il faut bien comprendre qu'un binôme, c'est 2 bras en action (comme un « simple » développeur) mais 2 cerveaux (la véritable valeur ajoutée). Pour tirer pleinement partie du binômage, il est donc important de valoriser « l'utilisation du cerveau » et des personnes : des tâches, projets et des processus de courte durée (itératifs & incrémentaux) pour favoriser la confrontation, la communication, la réflexion.

Enfin, le binômage est intensif (un rythme durable est donc conseillé pour envisager un bon fonctionnement sur le long terme) et contraignant ce qui nécessite des profils adaptés.

Cette citation d'un « grand penseur » contemporain traduit bien le véritable challenge du binômage :

« 1 + 1 = 1 [...] 1 + 1 = 11 » (Jean-Claude Van Damme).

Ressources

Le livre « Pair Programming Illuminated » (2002) de Laurie Williams est quelques extraits disponibles :

- Ch 3 : The Seven Synergistic Behaviors of Pair Programming
- Ch 4 : Overcoming Management Resistance to Pair Programming
- Pair Programming: Experience the Difference

Les publications de Laurie WILLIAMS :

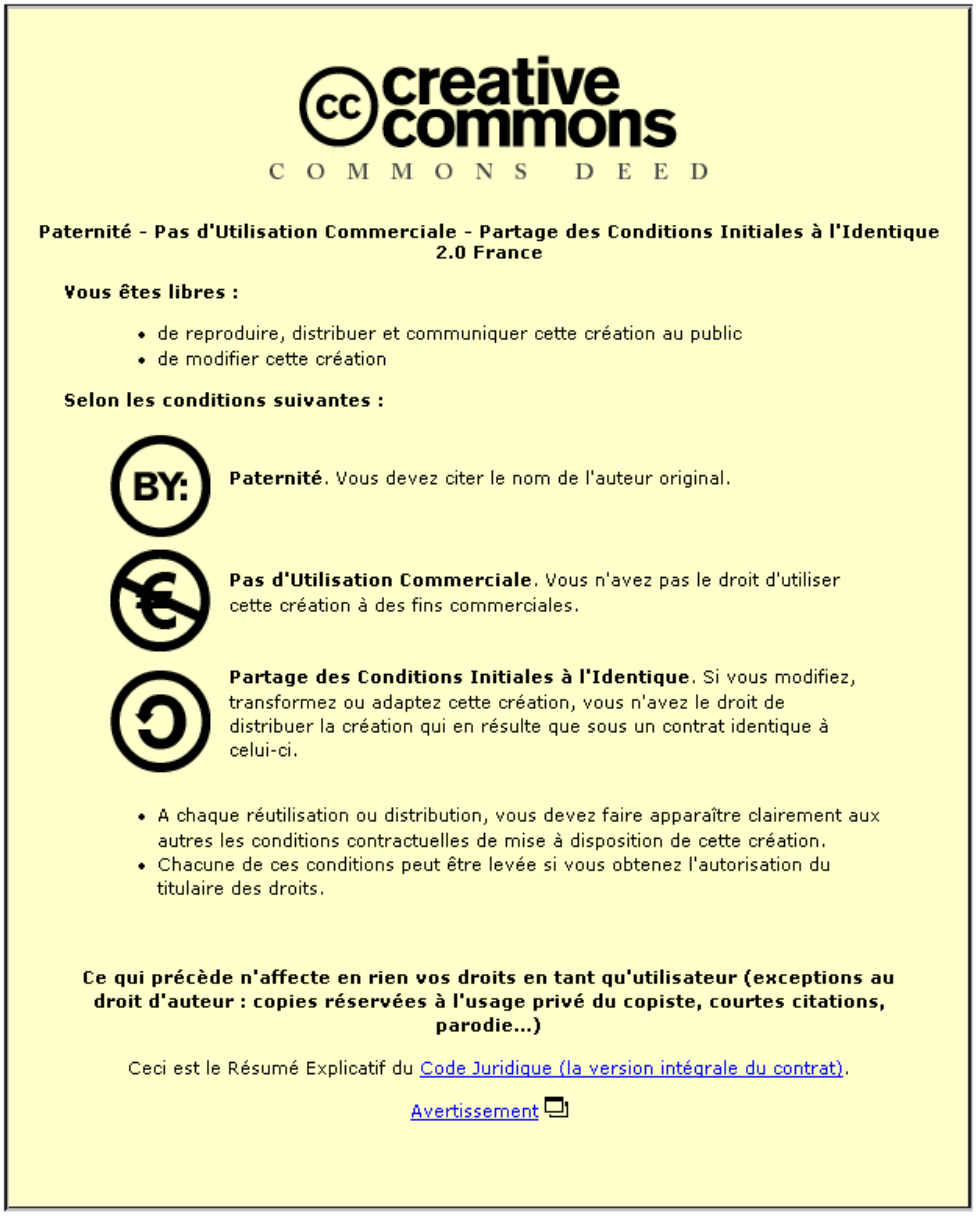
- Selling Pair Programming
- The Costs and Benefits of Pair Programming
- Strengthening the Case for Pair-Programming
- Pair Programming: Experience the Difference

A Pair Programming Experience

Don't restrict pair programming to difficult tasks, Dominic WILLIAMS

Contrat Creative Commons

Cette création est mise à disposition selon le « *Contrat Paternité-Pas d'Utilisation Commerciale-Partage des Conditions Initiales à l'Identique 2.0 France* » disponible en ligne http://creativecommons.org/licenses/by-nc-sa/2.0/fr/ ou par courrier postal à Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.



creative commons

COMMONS DEED

Paternité - Pas d'Utilisation Commerciale - Partage des Conditions Initiales à l'Identique 2.0 France

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public
- de modifier cette création

Selon les conditions suivantes :

BY: **Paternité**. Vous devez citer le nom de l'auteur original.

Pas d'Utilisation Commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

Partage des Conditions Initiales à l'Identique. Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.

- A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.
- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.

Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...)

Ceci est le Résumé Explicatif du Code Juridique (la version intégrale du contrat).

Avertissement