



# *Introduction*

## *Les bâtisseurs de cathédrale du XXI<sup>ème</sup> siècle*

---

Aujourd’hui, de nombreuses entreprises ont adopté les processus agiles tels que Scrum et Kanban (voir le TechTrends #3). Ces processus reposent sur une amélioration continue de la conduite de projet afin d’être toujours plus réactif face à un marché et une demande client très changeants.

Le constat est que trop souvent les projets agiles déçoivent. Ils se heurtent fréquemment à des problématiques que les méthodes agiles n’abordent pas assez : les pratiques de développement logiciel, la qualité intrinsèque des équipes et leur capacité à intégrer en continu de nouvelles méthodes de réalisation d’un logiciel de très haute qualité au juste coût.

Il est désespérant de constater que, quelle que soit la méthode choisie, la durée de vie des projets informatiques dépasse rarement cinq ans : code non maintenable, connaissance perdue au gré des départs des “développeurs clés”, technologies devenues obsolètes avant même la mise en production, etc.

Cet amer constat a poussé les premiers concernés, à savoir les équipes de développement, à agir. C'est ainsi qu'est né le mouvement Software Craftsmanship qui a énoncé son objectif dans un manifeste dès 2009 :

 As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft.

Traduit par la "maîtrise du métier", ce mouvement utilise la qualité et l'apprentissage comme clés de voûte d'une approche visant l'Etat de l'Art. Ainsi, il emprunte son esprit aux pratiques du compagnonnage de l'époque médiévale. Les cathédrales ont traversé les siècles car les corporations qui les ont bâties appliquaient des règles professionnelles très strictes qui trouvent un écho dans les piliers du Software Craftsmanship :

- **l'apprentissage** : l'apprenti devient compagnon puis maître développeur,
- **le mentoring** favorisé par l'aspect communautaire et certaines pratiques telles que le binômage et les revues de code,
- **la discipline** nécessaire à l'application des pratiques,
- **la passion** et l'amour du travail bien fait : Software Development Done Right.

Tout responsable informatique en a un jour fait l'expérience : un code de mauvaise qualité engendre un nombre important de bugs et de régressions et impacte fortement les coûts d'un projet (les coûts directs, bien sûr, mais aussi les coûts cachés : perte de chiffre d'affaires, altération de l'image de marque et faible évolutivité).

Nous sommes convaincus que les bénéfices du craftsmanship au sein de vos équipes sont réels : une base de code évolutive et à juste coût, une montée en compétences homogène de l'équipe et des engagements de développement en termes de délai et de qualité plus souvent respectés.

Les fondations de votre SI doivent être solides afin de supporter l'ensemble des fonctionnalités qui différencient votre entreprise. Colmater des brèches ponctuellement, à l'aide d'experts, est une stratégie à court terme qui vous mènera à coup sûr dans l'impasse. Voyez loin.

Ce TechTrends vous permettra d'identifier l'attitude type des software craftsmen, de comprendre comment ils travaillent et de faire émerger ces comportements et ces pratiques au sein de vos équipes, pour en retirer tous les bénéfices.



# Initier



## Nul doute que si Notre Dame de Paris

est aujourd’hui l’un des monuments les plus visités dans le monde, au delà même de sa nature de cathédrale, c’est sa durabilité dans le temps et le fait qu’elle soit parvenue jusqu’au XXIème siècle, qui fait sa grandeur.

Cette durabilité découle de :

- la qualité de la construction,
- la capacité de rénovation et d’amélioration à travers les âges.

Même si les durées ne sont pas comparables, l’analogie avec nos systèmes d’information est grande. Comment faire pour que les logiciels développés par mes équipes ne soient pas à jeter au bout de cinq ans ? Si mon métier se transforme (citons, par exemple, la révolution mobile et le “toujours connecté”), serais-je en mesure d’ajouter des briques à mon SI ou bien devrais-je tout revoir de fond en comble ?

De plus, de nombreuses innovations sont nées de ces chantiers titaniques, que ce soit au sein même du métier de la construction (la voûte en ogive, la cage à écureuil) ou bien dans des domaines connexes (le transport des matériaux par exemple).

Le milieu du développement logiciel est comparable. Des applications innovantes comme GMail sont nées de l’opiniâtreté d’équipes de développement passionnées.

## DÉFINISSEZ LES BASES DE LA CONSTRUCTION

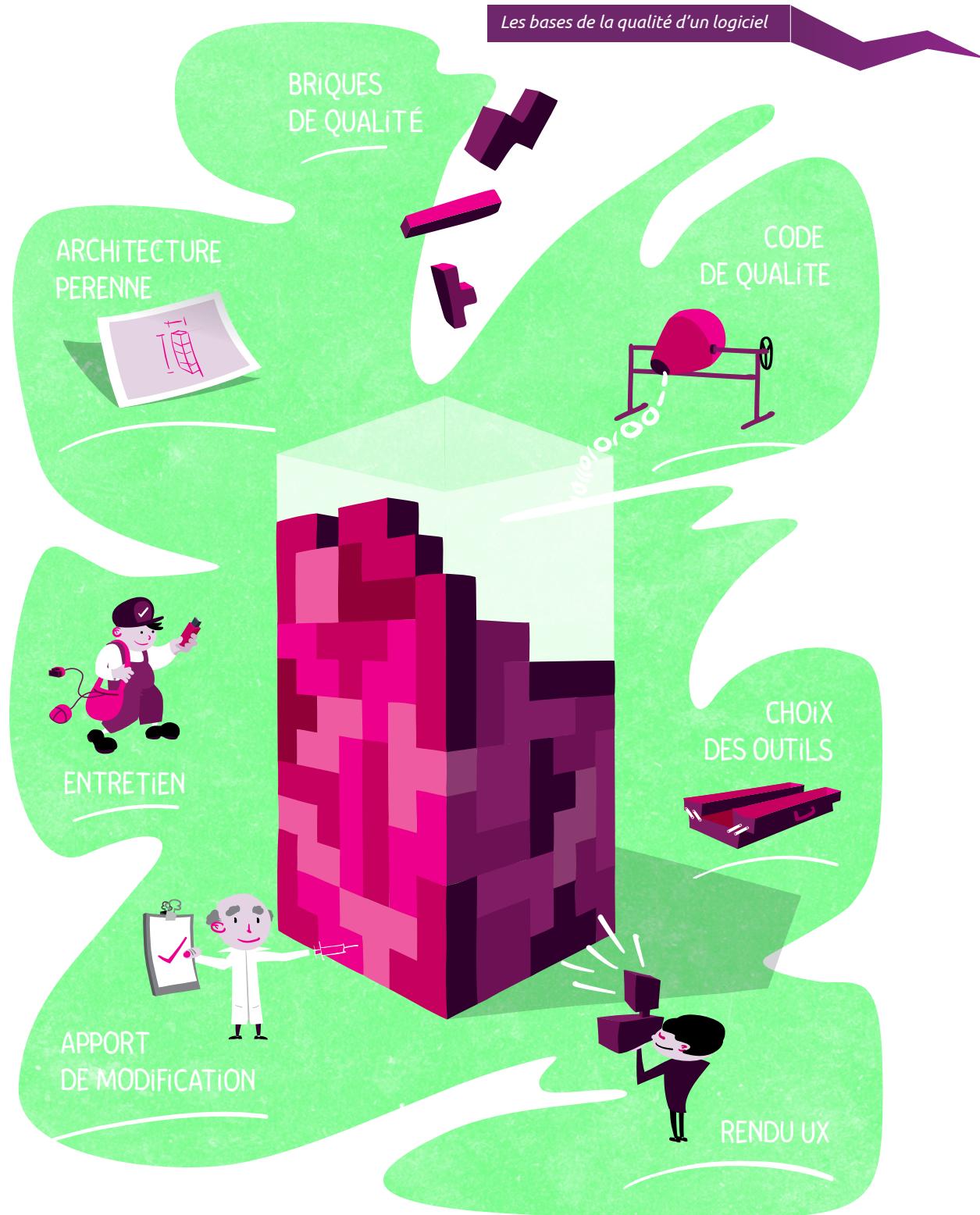
Les concepts ne sont pas si éloignés. La qualité d’un logiciel, comme celle d’une construction, repose sur :

- le choix d’une architecture pérenne et modulaire,
- la qualité des briques logicielles utilisées,
- la qualité intrinsèque du code qui permet de les assembler (le ciment),
- le choix des outils et le rendement qu’ils procurent aux développeurs,
- le rendu final aux yeux des utilisateurs,
- la facilité, à travers le temps, à apporter des modifications, qu’elles soient par petites touches cosmétiques ou bien plus en profondeur,
- le soin apporté à l’entretien.



*Si mon métier se transforme (citons, par exemple, la révolution mobile et le “toujours connecté”), serais-je en mesure d’ajouter des briques à mon SI ou bien devrais-je tout revoir de fond en comble ?*





La notion de qualité dépend du point de vue de celui qui observe :

- l'utilisateur final considérera que le logiciel est réussi s'il est adapté à ses besoins mais aussi, et surtout, s'il est capable de répondre rapidement à ses nouvelles attentes.
- Un DSI ou un chef de projet aura, lui, une vision plus opérationnelle : le projet sera considéré comme un succès si l'utilisateur final est satisfait dans des délais raisonnables et pour un budget maîtrisé.
- Le développeur, enfin, pourra être fier du travail produit, si les deux conditions précédentes sont remplies et si, de son point de vue, son code est lisible, stable, évolutif et pourra "lui survivre".

En prônant une qualité sans compromis, teinté d'un grand pragmatisme, le mouvement Software Craftsmanship permet d'atteindre ces objectifs. En poussant les artisans logiciel à améliorer de manière quotidienne leur production, à progresser en tant qu'équipe, les objectifs qualitatifs purement liés au développement seront atteints. Mais qu'en est-il de la satisfaction des objectifs opérationnels et fonctionnels ?

Par effet de bord, l'impact sur ces derniers en sera visible : un code de qualité permet d'éviter des dérives budgétaires et temporelles importantes. Que ce soit au niveau des défauts produits, de la rapidité de leur correction ou de l'ajout de nouvelles fonctionnalités, le coût de ces interventions sera plus facilement contrôlé. Si on maîtrise la durée de production et son budget, il devient alors plus facile d'accepter le changement et de produire rapidement des fonctionnalités en adéquation avec les nouvelles attentes des utilisateurs.

# ORGANISEZ VOTRE ÉQUIPE CRAFT

La réussite d'un projet titanique comme Notre Dame de Paris repose avant tout sur les hommes : les religieux qui ont "sponsorisé" le projet à la fois financièrement et en faisant la promotion, les architectes qui ont dessiné les plans et posé les bases de l'édifice, mais aussi et surtout, les générations successives d'artisans qui ont édifié la cathédrale, en apportant leur savoir-faire, leurs méthodes, leurs outils et parfois en contredisant les plans initiaux pour parvenir à une meilleure réalisation.

L'informatique suit les mêmes règles. Pour durer, compter systématiquement sur un super-héros est déraisonnable. Il est préférable de se reposer sur une équipe solide, soudée autour de valeurs clés :

## La passion

Développer est plus qu'un métier. L'amour du travail bien fait n'est pas un vain mot. La pérennité de son travail, la production d'un code toujours opérationnel dans cinq ans, sont des fiertés de tous les jours. Un craftsman est enclin à s'impliquer et s'investir en optimisant le code qu'il produit, en cherchant toujours la meilleure solution au problème et en repoussant encore plus loin sa connaissance. Sa passion est une source d'échange et d'inspiration pour l'ensemble de l'équipe.

En dehors de leur travail au jour le jour, les craftsmen sont des spectateurs attentifs des évolutions de leur métier, quand ils ne sont pas eux mêmes directement acteurs de la communauté. Par ailleurs, ces dernières années ont vu émerger de nombreuses réunions (les User Groups, Meetups ou Tech Events) qui attirent, en soirée, de nombreux développeurs autour d'une technologie ou d'une pratique donnée. Comme dans toutes les disciplines : ceux qui y excellent sont ceux qui pratiquent avec passion.

## L'implication

Produire des lignes de code n'est pas une finalité en soi. Votre équipe de craftsmen voudra s'impliquer dans le projet de A à Z, de la conception à la réalisation. Ne voyez pas cette volonté comme une dispersion mais prenez-la, au contraire, comme une force : un développeur sera toujours plus efficace s'il comprend les problématiques en amont et en aval, qu'elles soient fonctionnelles ou techniques.

Vos équipes de développement ne sont pas constituées de simples exécutants. La qualité de la modélisation technique de vos projets n'est pas un asset mineur pour votre SI, elle est primordiale (voir Domain Driven Design). La remise en cause de choix architecturaux, voire parfois fonctionnels, ne sont pas des affronts faits aux penseurs mais plutôt le reflet d'une volonté commune d'atteindre un niveau d'excellence. L'équipe devient alors force de proposition.

## Le pragmatisme et le bon sens

Un bon artisan ne perd jamais de vue le but de son travail : produire un logiciel de qualité, certes, mais répondant avant tout aux exigences de ses utilisateurs. Il est parfois tentant d'expérimenter la dernière technologie dont tout le monde parle. Or, l'objectif est de choisir les briques adaptées au contexte. Le choix doit être arbitré à la lumière du triptyque inhérent à chaque projet : coût, respect des délais, qualité. Les meilleurs artisans logiciels sont ceux qui sont capables de réaliser des choix pragmatiques et éclairés et non ceux qui suivent les règles de la « mode ».

## La légitimité et l'humilité

Les craftsmen resteront intransigeants sur la qualité et sauront dire "non" grâce à une légitimité acquise auprès de l'ensemble des parties prenantes du projet : Product Owner, Scrum Master, développeurs, équipes de production, etc.

La prise de conscience des enjeux métier et la communication sont les piliers de la relation de confiance que l'équipe craft doit maintenir avec les représentants fonctionnels. Démontrer qu'une feature repose sur du code de qualité, parfois plus long à produire qu'un patch sur un coin de table, n'est pas une sinécure. Sans une confiance mutuelle entre métier et développeur, le conflit d'opinion est assuré : il faut avant tout qu'une relation professionnelle de confiance s'installe entre tous les acteurs des projets.



*Un craftsman est enclin à s'impliquer et s'investir en optimisant le code qu'il produit, en cherchant toujours la meilleure solution au problème et en repoussant encore plus loin sa connaissance. Sa passion est une source d'échange et d'inspiration pour l'ensemble de l'équipe.*





Être un bon craftsman est un métier exigeant, qui demande une implication que certains jugeront parfois excessive. Il faut néanmoins raisonner en termes d'équipe, dans laquelle chacun amènera son énergie et ses compétences. Même si des disparités de niveau existent, un développeur ne s'inscrivant pas dans cette démarche aura du mal à trouver sa place, voire pénalisera même l'équipe dans sa productivité.

## Take away Craftsmanship Trends



### INITIER

- Définir les objectifs et aligner les visions de tous les participants du projet sur la notion de qualité logicielle :
  - Répondre aux besoins des utilisateurs,
  - Développer avec des coûts et des délais maîtrisés,
  - Présenter des qualités intrinsèques au projet : modularité, stabilité, évolutivité, lisibilité.
- Constituer et entretenir des équipes de craftsmen passionnés, impliqués, pragmatiques, humbles et légitimes.

# Construire



## Une fois les ambitions définies

(répondre aux besoins des utilisateurs, une qualité sans compromis, une équipe motivée), il n'y a plus qu'à bâtir. C'est un travail de longue haleine, qui demande une attention et une remise en cause quotidiennes.

Le mouvement craftsmanship amène un ensemble d'outils et de pratiques qui doivent vous assurer un chantier "tranquille", mais surtout d'éviter les dérives dans le temps.

## CRÉEZ UNE ÉQUIPE UNIFIÉE ET HOMOGÈNE

Lors de la construction des cathédrales, il était bien évident que tous les corps de métier se côtoyaient sur le même projet : forgerons, charpentiers, maçons, verriers, etc. Tous étaient spécialisés dans leur domaine de compétence, offrant une valeur ajoutée à leur présence sur le chantier. À l'heure actuelle, dans le monde informatique, la nécessité d'une spécialisation pour chacun des développeurs apparaît moins tranchée.

L'un des principaux écueils dans la construction de son équipe est de se reposer intégralement sur un "super héros". Que ce dernier possède la majorité de la compétence technique ou la plus large connaissance fonctionnelle de l'équipe, son départ, même temporaire, sclérosera l'équipe. Elle ne sera plus en mesure de délivrer la même valeur qu'à son habitude. La solution la plus simple pour palier à ce type de problème est de constituer des équipes pluridisciplinaires et homogènes techniquement. N'obligez pas un développeur à avoir un domaine réservé où il sera le seul spécialiste. Le but ultime étant que tous les membres de l'équipe soient complémentaires mais pas indispensables.

Il est bien évidemment très compliqué, voire impossible, d'atteindre un tel but, des disparités existent forcément. La difficulté consiste donc à les lisser en tirant l'équipe vers le haut. C'est l'un des principaux objectifs poursuivi par le Software Craftsmanship, qui promeut l'application de méthodes simples pour l'accomplir.

Il faut, tout d'abord, favoriser la cohésion. Une équipe solidaire sera plus forte que des individualités éparses. Il est nécessaire d'aménager des espaces de discussion, qu'ils soient réels ou virtuels, dans le temps et dans l'espace. Le must reste bien sûr d'avoir une équipe colocalisée. Malgré tout, il est vain de nier l'existence de l'offshoring. Dans ce cas, il est nécessaire de se doter d'outils (vidéo conférence, partage d'écrans, etc.) et de bonnes pratiques qui permettent de minimiser les impacts de l'éloignement. La solution la plus efficace est de faire voyager des membres des équipes pour des rencontres physiques. Il en résultera un meilleur esprit d'équipe grâce à une communication plus riche.

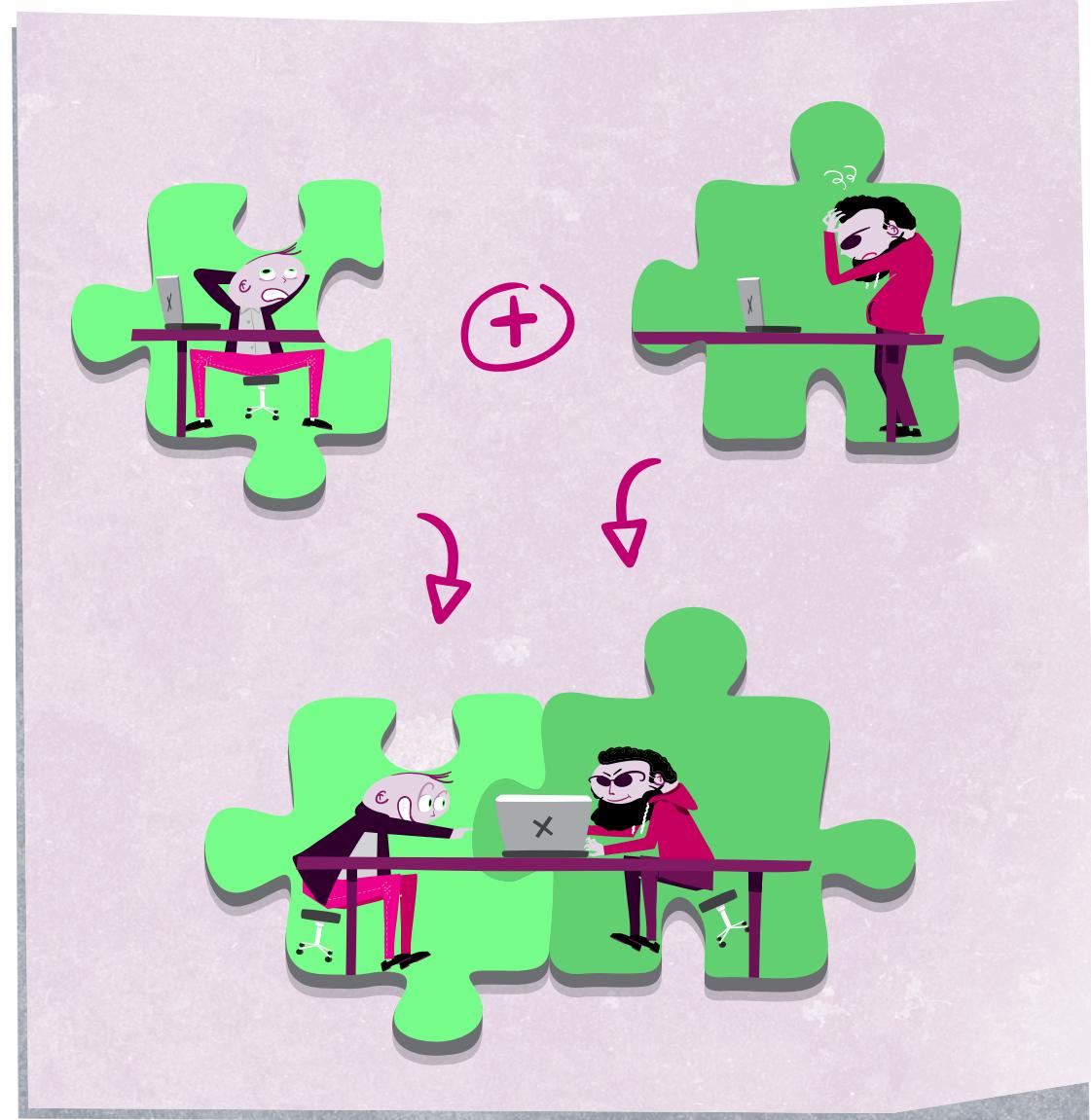
Les cérémonies agiles sont un bon point de départ :

- Le stand-up quotidien permet à l'équipe d'échanger tous les jours sur les éventuelles difficultés rencontrées et de prendre des décisions permettant d'améliorer l'efficacité, en proposant, par exemple, de participer à une session de pair programming ou de revue de code, pratiques crafts (voir ci-dessous).
- La rétrospective est également l'occasion de mettre en avant d'éventuelles malfaçons dans la manière de coder ou des difficultés à appréhender un composant du programme.

Vient ensuite la formation en interne. Au même titre que les maîtres tailleurs commencent au bas de l'échelle en tant que "lapin", les compagnons développeurs doivent apprendre des "maîtres" ayant une plus grande expérience et/ou une plus grande virtuosité. Cette évolution passe par l'observation et surtout la pratique. Dans le monde du développement logiciel, l'utilisation des méthodes de l'eXtreme Programming (XP) donne des résultats convaincants.

Une de ces méthodes, le pair programming, favorise grandement le partage de la connaissance au sein de l'équipe par l'utilisation d'une seule station de développement pour deux personnes. Cette mutualisation de deux cerveaux sur une même tâche peut apparaître comme une perte sèche et un gaspillage de ressources. Il n'en est rien :

- Le dialogue engagé entre les deux membres du binôme permet de mettre en lumière les difficultés inhérentes à la fonctionnalité développée et d'attaquer ces problèmes par deux angles de vues distincts.  
De ce brainstorming naît généralement une solution de plus haut niveau.
- La communication permanente entre les deux membres du binôme a aussi des effets indirects encore plus intéressants :
  - Les pratiques de développement de chacun sont exposées sur un cas réel. C'est la meilleure façon, pour chacun, d'apprendre de l'autre : il est fréquent, lors de session de pair programming, d'entendre demander quel raccourci a été utilisé ou bien pourquoi la séparation entre méthodes a été réalisée de telle façon.
  - La connaissance technique et métier est partagée de manière complètement transparente. Les deux développeurs seront ainsi à même d'expliquer ou d'intervenir ultérieurement sur la partie de code développée en commun.



Le pair programming



*Formations*

*Pair  
programming*

*Responsabilité  
collective*

*Revue  
de code*

*Expérience*

En complément à la pratique régulière du pair programming, l'équipe peut avoir recours à la revue de code : un équipier, plutôt expérimenté, relit une partie du code avec son auteur (ou avec l'équipe dans son ensemble), émet des critiques (constructives) et retravaille le code en conséquence. La réalisation fréquente de revue de code croisée favorise la mise à niveau du code et évite de laisser des zones inexplorées devenir une dangereuse gangrène pour l'ensemble du logiciel. Elle renforce également la notion de "responsabilité collective" de la base de code.

De la même façon qu'il est souhaitable de posséder une équipe mixte au niveau de l'expérience, il est tout aussi utile de localiser au sein de la même équipe toutes les compétences. Trop souvent, des fonctions n'étant pas considérées comme du développement pur, comme la User Experience et l'exploitation de la donnée, sont traitées dans des silos en amont ou en aval du développement. Pourtant, même si les métiers ne sont pas exactement identiques, les membres de l'équipe gagnent énormément à apprendre les uns des autres et trouvent souvent d'excellents compromis au bénéfice du projet.

---

## CHOISISSEZ LES BONS OUTILS

L'édification d'une cathédrale comme Notre Dame s'est étalée sur plusieurs dizaines d'années. Malgré tout, chaque artisan devait faire preuve d'une grande efficacité. Pour cela, chacun d'entre eux était armé de ses propres outils et méthodes de constructions : au tailleur de pierre ses ciseaux, au verrier son grugeoir et au ramoneur son furet. L'analogie est facile : vos artisans développeurs doivent être bien outillés !

### Façonner son code à l'aide d'un environnement de développement (IDE)

Au même titre qu'un tailleur de pierre qui arrive sur un chantier avec son choix de ciseaux, un développeur sera plus efficace s'il travaille avec ses propres outils, principalement son IDE. Il est fréquent que, pour des raisons de stratégie d'entreprise, le choix de l'IDE soit imposé à un niveau global. Ce choix, même s'il est raisonnable, peut rapidement s'avérer contre productif. Observez un développeur connaissant son IDE sur le bout des doigts : vous aurez parfois du mal à comprendre comment il peut manipuler si rapidement ses lignes de code. Il connaît simplement tous les raccourcis et méthodes qui lui facilitent la vie au quotidien. Forcez-le à prendre un autre IDE : il repartira de zéro et devra réapprendre tous les trucs et astuces.

La stratégie consistant à homogénéiser les IDE au sein de l'entreprise trouve ses racines dans plusieurs pratiques qui peuvent dorénavant sembler obsolètes :

- Le coût de licence. Aujourd'hui, la majorité des IDE propose des versions Open Source plus qu'acceptables. Le coût ne devrait donc plus être un problème mais si l'un de vos développeurs demande une version payante de son IDE, peut-être cela vaut-il la peine de le challenger sur ses réels apports. Souvent, le coût des licences par rapport aux gains de productivité escomptés reste dérisoire.
- L'existence de plugin, de règles de code, de frameworks spécifiques à l'entreprise. Nous n'allons pas ici rentrer dans un long débat, mais l'expérience a montré que se reposer sur les standards du marché est, à long terme, un meilleur pari que le re-développement de solutions "maison".

### Utiliser des librairies éprouvées

La force d'un craftsman repose également en grande partie sur la connaissance des usages et techniques de son métier. Dans le cas de l'industrie du développement logiciel, il doit faire face à un réel foisonnement d'outils et de librairies tierces. La veille technologique est donc un devoir : le craftsman doit savoir sur quelles briques il peut se reposer pour ne pas avoir à réinventer la roue, tout en gardant un contrôle absolu sur son projet.

De nombreuses librairies permettent de compléter la grammaire "basique" des langages de développement en leur offrant une meilleure expressivité, une plus grande productivité ou tout simplement en ajoutant des fonctions de plus haut niveau.

Pour illustrer notre propos, citons quelques acteurs établis : Spring, Hibernate, JUnit, Mockito, et quelques nouveaux entrants comme Dagger, Spoke, ou Scalatest. Nous n'allons pas ici faire une liste exhaustive des utilisations de ces librairies, qui sont en grande partie dépendantes des caractéristiques de l'entreprise et du projet.



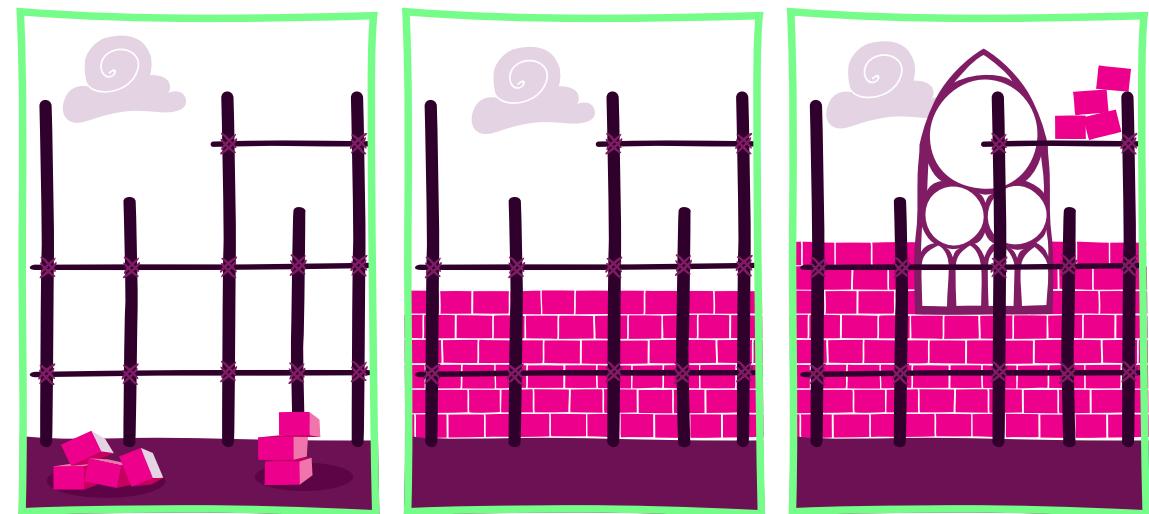
*La veille technologique est donc un devoir : le craftsman doit savoir sur quelles briques il peut se reposer pour ne pas avoir à réinventer la roue, tout en gardant un contrôle absolu sur son projet.*



## TESTEZ AVANT DE CONSTRUIRE

La voûte à ogive de Notre Dame de 33 mètres que nous connaissons actuellement n'a pas été érigée en une fois. Les artisans ont validé leurs méthodes de construction petit à petit. Ils se sont ensuite appuyés sur des échafaudages solides et en adéquation avec la vision initiale, pour construire une structure à l'image de ce qu'ils souhaitaient.

La pratique du "test-first", prônée par le mouvement craftsmanship, découle du même principe. Coder un test unitaire qui décrit le comportement du code que l'on veut rédiger (quels paramètres prend-il en entrée et quel résultat dois-je obtenir ?) est une façon de garantir que le code produit par la suite sera de qualité et cohérent avec les attentes.



Test First au Moyen Âge

Coder les tests en premier favorise un design émergent plutôt que la prévision de la totalité des mécanismes qui permettront d'articuler l'ensemble du code des applications. Les développeurs agissent de manière locale et laissent ces zones d'articulation apparaître naturellement. Cela permet d'adhérer facilement à un design "Clean Code" : Keep It Simple and Stupid et You Ain't Gonna Need It. Les développeurs ne développent que ce qui est nécessaire pour répondre au besoin, en évitant le code superflu qui représente un coût.

L'autre bénéfice, non négligeable, à posséder des tests unitaires (qu'ils aient été écrits avant le code ou après) est de disposer d'un filet de sécurité permettant d'envisager sereinement la suite des développements : si uniairement chaque méthode réagit comme je l'ai "spécifié" alors la combinaison de toutes les méthodes de mon code devrait continuer à fonctionner de manière iso. Les anomalies introduites seront détectées plus rapidement et leur coût de correction grandement diminué.

En parallèle, la technique du refactoring, qui consiste à modifier l'agencement technique d'une partie du code en conservant ses fonctionnalités, aura une assurance "fonctionnement garanti" sur laquelle s'appuyer.

Parmi les autres avantages d'avoir un code suffisamment et correctement testé, citons également :

- La documentation du code par ses tests.
- La facilité pour un nouvel entrant de comprendre le fonctionnement du code, en l'observant du point de vue de ses fonctionnalités (le test unitaire documente également l'utilisation du code qu'il couvre).
- Un meilleur découpage du code source : si du code est difficile à tester, c'est probablement qu'il est mal pensé et qu'il est temps de le refactorer.

Les équipes de craftsmen ont systématisé cette approche, à travers la pratique du Test Driven Development (TDD) : l'ajout de chaque nouvelle fonctionnalité est piloté par les tests, en suivant un principe Red / Green / Refactor :

1. **Red** : le développeur écrit un test automatisé qui teste le comportement attendu de la nouvelle fonctionnalité. Dans un premier temps, ce test doit échouer, car le code permettant de le satisfaire n'est pas encore rédigé.
2. **Green** : le développeur écrit le minimum de code nécessaire permettant au test précédent d'être exécuté avec succès.
3. **Refactor** : le développeur améliore le code précédemment écrit tout en maintenant les tests en succès.

Ce principe est aussi applicable lors de la correction d'anomalies : un test reproduit le cas d'erreur avant correction. Ce test garantit une correction effective, l'absence de nouvelles régressions et enfin que cette anomalie ne ressurgira pas lors d'un développement ultérieur.

L'utilisation d'un système d'intégration continue, qui exécute périodiquement l'ensemble des tests, est un plus indéniable permettant de détecter les régressions au plus tôt et de sensibiliser les développeurs à la qualité.

Même si la mise en place d'outils d'intégration continue n'est pas directement liée au mouvement Craftsmanship (on a plutôt tendance à les rapprocher du mouvement DevOps, voir le TechTrends #2), les craftsmen sont demandeurs de tels outils. En effet, ceux-ci leur apportent un feedback rapide et précis de l'état du code à tout instant. Il est même très fréquent de voir les équipes de développement demander un moniteur spécifique qui affiche, en temps réel, la qualité des développements réalisés. Conscient qu'il construit un produit qui devra être exploité, le craftsman aura soin de travailler aussi tôt que possible dans un environnement proche de celui de la production.

## UTILISEZ UN LANGAGE COMMUN À TOUS

### Le Behavior Driven Development (BDD)

En complément de la pratique du TDD, nous recommandons l'utilisation du Behavior Driven Development (BDD). Cette méthode vise à faire travailler en étroite collaboration le Product Owner et l'équipe de développement, afin de rédiger les stories et le code test associé, avec une écriture commune à tous. Cette dernière est basée sur le comportement attendu de l'application via 3 injonctions simples :

- "Etant donné" (Given) : décrit les conditions initiales du test,
- "Quand" (When) : décrit l'opération réalisée,
- "Alors" (Then) : décrit le résultat attendu.

Cette grammaire a l'avantage de proposer un découpage systématique et reproductible des tests. En s'appuyant sur l'usine d'intégration continue, la spécification sera exécutée à chaque activité sur le projet permettant un retour immédiat. Elle teste le code en continu et fournit un état clair et non équivoque sur l'avancement d'une feature dans un projet. De plus, ces dernières sont alors versionnées avec le code, assurant une meilleure traçabilité.

D'autres avantages de la pratique du BDD sont rapidement visibles : l'accentuation des échanges entre tous les participants d'un projet, qu'ils soient techniques ou fonctionnels, la fluidité des processus de décision par une compréhension mutuelle des attentes et enfin la cohérence entre les objectifs de réalisation du code et son bénéfice final.

## Le Domain Driven Design (DDD)

En complément, nous prônons la pratique du Domain Driven Design (DDD) visant la mise en place d'un "Ubiquitous Language". En effet, lors du développement logiciel, il se pose le problème de l'utilisation de plusieurs langages : l'un pour les spécifications, l'autre pour le code de test et enfin un troisième pour le code de production. Par exemple, il n'est pas rare que l'objet "prix" soit aussi cité par les termes "tarifs" ou encore "frais" dans les différents artefacts du projet. Afin d'homogénéiser le vocabulaire, chaque point spécifique d'une application possédera un nom unique déterminé par le Product Owner et sera connu de tous.

Dans un premier temps, cette homogénéisation permet de rédiger plus rapidement de nouvelles fonctionnalités mais aussi de rendre plus efficace la résolution de bugs "fonctionnels" car le code est exprimé dans un langage explicite.

Dans un second temps, c'est aussi la vision complète du projet qui est lissée et compréhensible par tout un chacun. Les transitions entre les 3 étapes clés du développement logiciel (spécification, test et code) ne seront alors plus un obstacle.

En fonction de l'avancée du projet et du code (legacy ou nouveau), l'ordre de mise en place de TDD, BDD et DDD sera différent. On réservera une approche TDD pour les nouvelles fonctionnalités. Le code existant, s'il n'était pas testé, est souvent difficile à couvrir. Mieux vaut adopter alors BDD, qui offre une vue plus systémique. DDD est complémentaire. Cela supprimera bien des obstacles dans le processus d'écriture des spécifications et du code.



*D'autres avantages de la pratique du BDD sont rapidement visibles : l'accentuation des échanges entre tous les participants d'un projet, qu'ils soient techniques ou fonctionnels, la fluidité des processus de décision par une compréhension mutuelle des attentes et enfin la cohérence entre les objectifs de réalisation du code et son bénéfice final.*



## PARTAGEZ LES RESPONSABILITÉS ET LA GLOIRE

Vous l'avez sûrement compris : nous avons essayé de systématiquement mettre en avant l'équipe plutôt que l'individu. En effet, la réalisation d'un logiciel est une entreprise collective. Le code doit être une base partagée et maintenue par tous. L'ego n'est pas le bienvenu, le statut de héros est votre ennemi ! Ce qui est vrai pour le code l'est aussi pour l'architecture, en raisonnant ensemble, l'équipe mûrit sa conception et permet à la meilleure architecture technique d'émerger par elle-même.

Le Collective Code Ownership doit devenir le mot d'ordre de vos équipes.

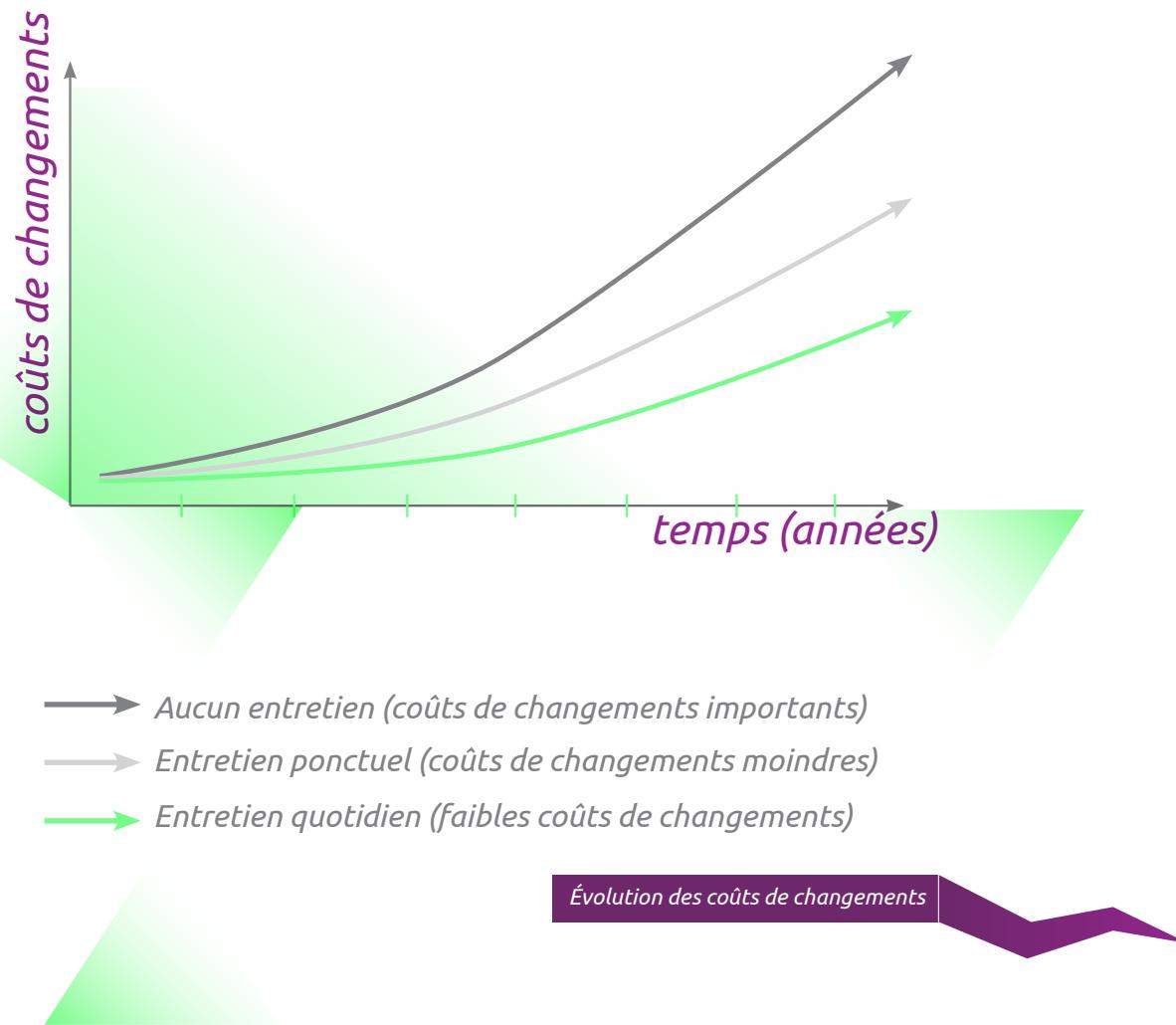
Les craftsmen, en utilisant le pair programming, le test-first, la communication poussée, s'affranchissent des limites individuelles et transforment chaque bloc de code en asset maîtrisé par l'ensemble de votre équipe. Chaque partie du système n'ayant aucune zone d'ombre pour eux, ils pourront être autonomes et réactifs sur l'ensemble du logiciel, même en l'absence du référent en la matière.

## CONSERVEZ UNE BASE SAINTE

Néanmoins, malgré toutes les bonnes intentions du monde et la discipline dont font preuve les développeurs, il ne faut pas se leurrer, tôt ou tard viendra un moment où, par manque de temps ou de communication, une anomalie "qualitative" sera introduite. Le tout est d'en avoir conscience et de la contrôler pour garder la maîtrise de l'ensemble.

Par exemple, lors de la construction de Notre Dame, il a parfois été nécessaire de bâtir des murs temporaires pour pouvoir accéder aux étages supérieurs, tout en gardant à l'esprit que ces murs ne pourraient jamais supporter le poids des feuilles de plomb de la toiture.

L'industrie logicielle présente ici encore des similitudes : il est parfois nécessaire d'aller au plus court, pour délivrer une feature dans les temps ou corriger un bug urgent. Cependant, ces "pansements sur une jambe de bois" doivent toujours être retravaillés, puis supprimés, au risque de voir la base de code devenir illisible, incompréhensible et donc non maintenable. On parle, dans le jargon informatique, de dette technique. Plus la dette est grande, plus le coût de développement est élevé. Le mouvement craftsmanship offre un ensemble d'outils qui donnent les moyens de maîtriser facilement cette dette.



La principale méthode de suppression de ces lignes de code qui se doivent d'être temporaires est le refactoring, que nous avons cité précédemment. C'est une activité que les artisans codeurs se doivent de pratiquer régulièrement, pour qu'elle devienne chez eux une seconde nature. Cette pratique est néanmoins risquée : il arrive, parfois, qu'en essayant de bien faire, on aggrave la situation. C'est pour cette raison que, comme toute technique maîtrisée, celle-ci s'accompagne d'un certain nombres de pratiques et de garde-fous (tests unitaires, maîtrise de l'IDE, découpage objet) que tous les membres de l'équipe craft se doivent de maîtriser.

## FAITES ÉVOLUER

La vie d'un projet informatique s'étale sur plusieurs années. De nombreux changements peuvent ainsi survenir au cours du temps : l'augmentation du périmètre fonctionnel, un changement profond du métier ou encore l'évolution des outils / frameworks choisis par l'équipe de développement. L'existence d'une base de code saine, évolutive et testée permet d'envisager sereinement toutes les modifications qui devront être apportées.

Par exemple, craindre de changer la version d'un framework afin d'intégrer des évolutions correspondant souvent à des améliorations opérationnelles (performance, stabilité, sécurité) est révélateur de l'état de fébrilité d'un projet. Une réelle interrogation quant à sa pérennité doit se poser.

Nous recommandons de suivre les roadmaps des composants de votre application et de procéder par petits incrément. Plus une tâche est réalisée souvent, moins elle apparaît difficile et risquée.



*La vie d'un projet informatique s'étale sur plusieurs années. De nombreux changements peuvent ainsi survenir au cours du temps : l'augmentation du périmètre fonctionnel, un changement profond du métier ou encore l'évolution des outils / frameworks choisis par l'équipe de développement. L'existence d'une base de code saine, évolutive et testée permet d'envisager sereinement toutes les modifications qui devront être apportées.*



## *Take away Craftsmanship Trends*



### **CONSTRUIRE**

- Favoriser la communication et les pratiques de pairing afin de maintenir l'homogénéité de l'équipe.
- Laisser les artisans choisir leurs outils, dans la mesure du possible et dans le respect des contraintes de l'entreprise.
- Encourager un environnement propice à la réalisation de tests, au maintien et au nettoyage (refactoring) fréquent du code.

# Entretenir



## Afin d'avoir des réalisations de qualité

toujours en adéquation avec les attentes des utilisateurs, il faut innover davantage et toujours plus vite. Pour suivre la cadence, l'innovation doit aussi se trouver au niveau des méthodes et outils utilisés par vos équipes.

Au Moyen-âge, la diffusion du savoir au sein d'un corps de métier se faisait principalement au travers des guildes qui regroupaient les artisans et leur permettaient d'échanger leurs savoir-faire.

Aujourd'hui, la guilde des développeurs n'a pas d'existence à proprement parler, mais les Software Craftsmen ont développé un réseau et des outils leur permettant d'être toujours dans les sphères d'excellence.

## PARTAGEZ LA CONNAISSANCE EN INTERNE

Nous avons précédemment vu qu'un premier niveau de partage peut être atteint via la pratique des méthodes XP. Il reste cependant souvent limité à quelques individus. Afin de garantir la cohésion de l'équipe au sens global (y compris les développeurs de l'équipe offshore), il est préférable de favoriser des échanges fréquents (tous les trimestres par exemple) mais aussi diversifiés (les équipes "métropolitaines" doivent aussi se déplacer vers les équipes near ou offshore). L'investissement paraît important, mais la cohésion et donc la qualité globale de l'équipe en sera renforcée.

### Institutionnaliser des sessions d'échanges

Le partage de la connaissance est une vieille chimère au sein des entreprises françaises. Faute de moyens nécessaires, les sessions de partage ressemblent parfois à une sinécure : proposées en fin de soirée, avec des ordres du jour bâclés à la dernière minute, les participants n'en retirent que peu de choses et cela tourne rapidement à la corvée.

Si vous décidez que le partage des connaissances est un vrai plus pour vos équipes de développement, mettez-y les moyens. Chez Xebia, nous avons instauré une journée mensuelle de partage de la connaissance depuis sa création. Les bénéfices induits par cet investissement conséquent sont bien supérieurs au coût de l'effort consenti.

### Accentuer les échanges en dehors du temps de travail

La pause de midi est une période favorable à l'échange. Si vous possédez un espace adapté (salle de réunion équipée d'un vidéo projecteur par exemple), mettez-le à disposition de vos équipes de développement. De nombreuses pratiques issues du mouvement craftsmanship y trouveront un lieu propice :

- Les Brown Bag Lunches (BBL). Ces sessions, au format conférence ou table ronde, voient généralement la présence d'un speaker désigné, interne ou externe à l'entreprise, qui présentera, pendant 1 à 2 heures, un sujet qu'il maîtrise. Le format BBL favorise les échanges et les participants doivent ressortir de ces conférences avec une vision d'ensemble de la technologie présentée et, éventuellement, avoir identifié des champs d'application au sein de l'entreprise et de leurs projets. Ce format permet une discussion "universitaire" mais aussi de profiter des retours de terrain.

• Les coding dojos. C'est en forgeant qu'on devient forgeron. L'adage est transposable au monde du génie logiciel. C'est en pratiquant le code que le jeune développeur s'aguerrira et deviendra "maître" développeur. Cela n'est pas toujours possible sur un projet, car il est impossible d'explorer au quotidien toutes les facettes du métier de développeur. Les séances de coding-dojo sont un moyen pour contourner ces problèmes : un groupe de personnes s'entraînent en binômes, sur un même problème durant 30 minutes. A la fin du temps imparti, le code est effacé, puis on recommence à coder avec un autre partenaire de travail. Le but est ainsi de se perfectionner et de découvrir de nouvelles techniques de conception ou de programmation. Ce simple moment de partage de connaissances peut s'avérer très bénéfique en marge d'un projet.



*La pause de midi est une période favorable à l'échange. Si vous possédez un espace adapté (salle de réunion équipée d'un vidéo projecteur par exemple), mettez-le à disposition de vos équipes de développement. De nombreuses pratiques issues du mouvement craftsmanship y trouveront un lieu propice.*



*Refactoring*

*Test first*

*Clean code*

*BDD*

*DDD*

## Offrir un temps dédié à la veille

Certaines entreprises vont plus loin, en dédiant du temps à la veille technologique de leurs effectifs. Ce temps est un réel investissement qui est compensé par l'acquisition et le renforcement des compétences de chacun des développeurs. Ce temps de veille peut être collectif (matinée mensuelle dédiée à un sujet émergeant en particulier) ou personnel (mise à niveau technologique, découverte du dernier framework à la mode). Ce temps de veille ne doit pas être considéré comme un dû par les développeurs. Ils doivent, au contraire, prouver à l'entreprise que l'investissement qu'elle consent à réaliser est une réelle opportunité. Chaque développeur devrait être en mesure de lister ses nouveaux acquis à la fin de chacune de ces périodes de veille.

Progresser et se maintenir à niveau est avant tout une démarche personnelle. De nombreux groupes de réflexion et outils numériques sont aujourd'hui disponibles pour le Craftsman.

## ÉCHANGEZ AVEC LA COMMUNAUTÉ

### Les User Groups, Meetups et Tech Events

Ces dix dernières années ont vues l'apparition de nombreuses guildes modernes, les "User Groups". Ces groupes de travail réunissent périodiquement des utilisateurs passionnés autour d'une même thématique technologique : les développeurs mobiles Android, les précurseurs utilisant Scala, etc. Ces groupes, la plupart du temps gratuits, sont ouverts à tous et permettent des sessions d'échange riches et vivantes. Qu'il soit simple spectateur ou bien acteur de ces communautés, le craftsman pourra continuer à apprendre au contact de ses pairs. Il portera souvent la bonne parole au sein de l'équipe dès le lendemain.



*Progresser et se maintenir à niveau est avant tout une démarche personnelle. De nombreux groupes de réflexion et outils numériques sont aujourd'hui disponibles pour le Craftsman.*



### Les conférences

Au delà de ces réunions mensuelles, la communauté s'est aussi organisée sur une échelle plus large en créant des conférences annuelles partout en France (Devoxx Paris, Mix-It, BreizhCamp) et en Europe (Devoxx Anvers, GeeCon, Jax). Ces conférences réunissent des milliers de passionnés, sur un ou plusieurs jours et abordent des thèmes variés. Elles permettent de cumuler, à grande échelle, tous les bienfaits de la veille technologique : aborder et défricher de nouveaux sujets, confronter sa vision avec une vision extérieure, se bâtir un réseau de "sachants", apprendre des retours terrain sur la mise en place des nouvelles technologies. Leur coût n'est pas négligeable pour un "simple développeur". La prise en charge financière par l'entreprise est une belle récompense pour le salarié mais aussi un investissement sur la connaissance qui est souvent payant pour l'entreprise.

### Les MOOCs

Autrefois plébiscité par des étudiants de tout âge, l'Université de Tous les Savoirs (ie des cours universitaires gratuits et accessibles à tous) se retrouve aujourd'hui massivement en ligne, à travers les Massive Open Online Courses. Pour une fois, les cordonniers ne sont pas les plus mal chaussés car une très grande partie de ces cours en ligne traitent d'informatique. La formation professionnelle n'est plus exclusivement payante et présente et dépend, là encore, de tout un chacun et non plus de décisions managériales. Si vos développeurs ont la volonté d'apprendre, il y a forcément une ressource en ligne qui leur permettra d'assouvir leur curiosité.

De nombreux professionnels justifient aujourd'hui d'une vraie expertise technologique sur des sujets qu'ils ont découvert par eux même, devant une vidéo de Coursera (pour ne citer que le leader de ce marché).

Cela va même plus loin. Si votre équipe est motivée, c'est tous ensemble qu'ils suivront le cours et travailleront sur les exercices d'approfondissement, créant ainsi une saine émulation.

Ces MOOCs sont aussi une chance pour les décideurs de se mettre régulièrement à niveau. En effet, très souvent, les premières semaines de cours offrent un survol technique et business permettant d'appréhender les opportunités de mise en place des technologies les plus récentes.

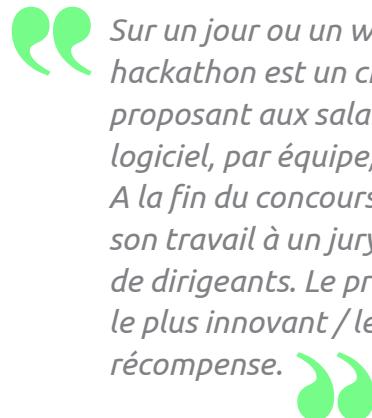
## Les hackathons

Sur un jour ou un week-end entier, un hackathon est un challenge d'innovation proposant aux salariés de développer un logiciel, par équipe, dans un délai imparti. A la fin du concours, chaque équipe présente son travail à un jury, constitué le plus souvent de dirigeants. Le produit le plus convainquant / le plus innovant / le plus fun aura alors une récompense.

Les hackathons sont un autre moyen trouvé par les entreprises pour faire coder leur salariés, en dehors de leurs projets quotidiens.

Ces événements offrent l'occasion de progresser collectivement via divers biais :

- Renouveler les équipes : de par leur format, les hackathons proposent souvent de faire collaborer des personnes qui d'ordinaire se trouvent dans des pôles différents. C'est une bonne occasion de favoriser la mixité.
- Proposer un cadre plus libre : généralement, la seule règle imposée par le hackathon est le thème. Le choix des frameworks de développement, des sujets précis, sont laissés aux équipes. Il s'agit souvent d'une belle occasion pour expérimenter de nouvelles technologies, voire mettre en avant des idées innovantes d'un point de vue fonctionnel.
- Faire émerger des produits et des cas d'usages innovants : c'est souvent le but recherché par les hackathons.



*Sur un jour ou un week-end entier, un hackathon est un challenge d'innovation proposant aux salariés de développer un logiciel, par équipe, dans un délai imparti. A la fin du concours, chaque équipe présente son travail à un jury, constitué le plus souvent de dirigeants. Le produit le plus convainquant / le plus innovant / le plus fun aura alors une récompense.*

## Take away Craftsmanship Trends



## ENTREtenir

- Aménager des moments d'échanges techniques et méthodologiques à des horaires raisonnables, pour que chaque membre de l'équipe puisse s'enrichir du savoir de l'autre.
- Miser sur la communauté pour ne pas rester en marge des innovations.

# Conclusion

Il est évident que les compétences globales d'une équipe de développement sont constituées de la somme des compétences des individus qui la composent.

Le mouvement Software Craftsmanship propose néanmoins un certain nombre de principes et de techniques qui permettent de gommer les individualités et de revenir à une abstraction de plus haut niveau : l'équipe.

Sa cohésion, sa passion et sa volonté de progresser collectivement sont, pour un directeur de projet, l'assurance que l'absence ou l'erreur d'un individu ne mettra pas en péril tout un projet.

De plus, les grands principes qui président à la rédaction du code source et aux choix des outils doivent permettre à chaque nouveau "compagnon" de s'approprier rapidement le projet et de garantir que ses premiers pas se feront sans crainte et sans mal, ni pour lui, ni pour la solidité globale du logiciel.

Enfin, vous ne pourrez peut-être pas attirer les meilleurs spécialistes dans chaque domaine mais nous vivons à une époque où le savoir est accessible et son partage une vraie volonté. Les communautés technologiques sont nombreuses et il ne faut pas hésiter à les utiliser pour progresser individuellement, puis en faire profiter le groupe.

Obtenir un logiciel de qualité, délivré par des passionnés, est la première étape de tout projet pérenne. Encadrez ces pratiques d'ingénieries logicielles par une méthodologie de définition de besoin pertinente en amont (Agilité, Lean Startup) et par de bonnes pratiques d'exploitation en aval (DevOps) : vous tutoierez peut-être la perfection atteinte par les batisseurs du XI<sup>e</sup> siècle.

## Take away Craftsmanship Trends



### INITIER

- Définir les objectifs et aligner les visions de tous les participants du projet sur la notion de qualité logicielle :
  - Répondre aux besoins des utilisateurs,
  - Développer avec des coûts et des délais maîtrisés,
  - Présenter des qualités intrinsèques au projet : modularité, stabilité, évolutivité, lisibilité.
- Constituer et entretenir des équipes de craftsmen passionnés, impliqués, pragmatiques, humbles et légitimes.



### CONSTRUIRE

- Favoriser la communication et les pratiques de pairing afin de maintenir l'homogénéité de l'équipe.
- Laisser les artisans choisir leurs outils, dans la mesure du possible et dans le respect des contraintes de l'entreprise.
- Encourager un environnement propice à la réalisation de tests, au maintien et au nettoyage (refactoring) fréquent du code.



### ENTRETIEN

- Aménager des moments d'échanges techniques et méthodologiques à des horaires raisonnables, pour que chaque membre de l'équipe puisse s'enrichir du savoir de l'autre.
- Miser sur la communauté pour ne pas rester en marge des innovations.

# Bibliographie



- Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 2008
- Andrew Hunt et David Thomas, *The Pragmatic Programmer: From Journeyman to Master*, 1999
- Kent Beck et Cynthia Andres, *Extreme Programming Explained: Embrace Change - 2<sup>nd</sup> Edition*, 2004
- Sandro Mancuso, *Software Craftsmanship Professionalism Pragmatism Pride*, 2014

## *Merci à*

*Audrey Pedro, Wendy Berton, Nicolas Jozwiak,  
Laurent Valdès, Clément Rochas, Marine Krôl  
et Marina Tracco*