TechTrends

Software Craftsmanship, Culture et pratiques

Initier, Construire, Entretenir

BY ENGINEERS *at* PUBLICIS SAPIENT

publicis
sapient

Les TechTrends sont l'expression de notre savoir-faire ; forgé sur le terrain, auprès de nos clients dans le cadre des projets que nous menons avec eux.

Fruit d'un travail collaboratif de nos consultants, vous y trouverez, nous l'espérons, les nouvelles tendances technologiques et méthodologiques ainsi que l'état de l'art de notre profession.

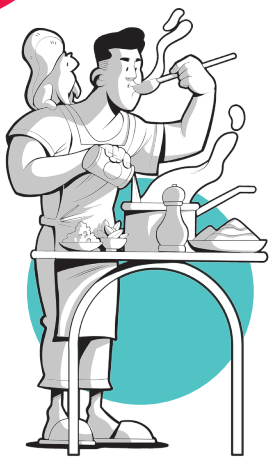
Nous tentons, dans le cadre de ces publications, de vous dispenser des conseils directement opérationnels afin de vous guider dans les décisions stratégiques que vous avez à prendre.

Distribués à plusieurs milliers d'exemplaires tous les ans, la collection des TechTrends s'étoffe régulièrement de nouveaux ouvrages.

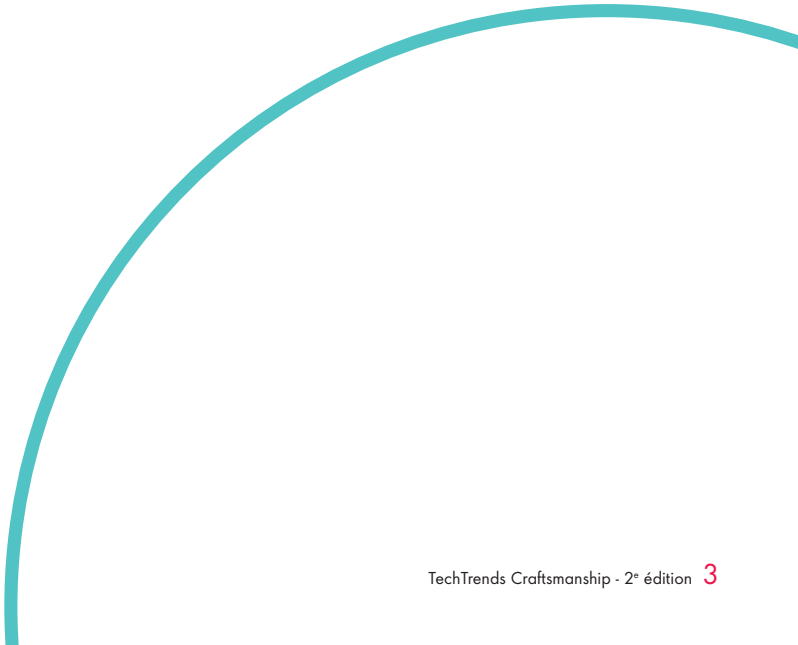
Bonne lecture !

Initier

Construire



Entretenir



Introduction

Aujourd'hui, de nombreuses entreprises ont adopté des méthodes et frameworks agiles tels que Scrum et Kanban (voir le TechTrends Agilité). Ces processus reposent sur une amélioration continue du fonctionnement des organisations afin d'être toujours plus réactif face à un marché et des besoins client très changeants.

Force est de constater que, malgré les transformations agiles, 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.

Cette « gueule de bois Agile », selon les termes de Sandro Mancuso [MANCUSO], est souvent due à un manque d'attention à l'excellence technique pendant les transformations.

Elle 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 [MANIFESTO] :

En tant qu'aspirants Artisans du Logiciel, nous relevons le niveau du développement professionnel de logiciels par la pratique et en aidant les autres à acquérir le savoir-faire[1]

Traduit par « L'artisanat du développement logiciel », ce mouvement utilise la qualité et l'apprentissage continu comme clé de voûte d'une approche visant l'état de l'art. Ainsi, il emprunte son esprit aux guildes et aux pratiques du compagnonnage de l'époque médiévale 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 le pair programming, le mob programming ou encore 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).

1- As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft : [...] référence disponible dans la bibliographie

Nous sommes convaincus que les bénéfices du Craftsmanship pour les équipes sont réels :

- une base de code évolutive et à juste coût,
- une montée en compétences homogène de l'équipe,
- une collaboration fructueuse entre les équipes de développement et leurs clients.

Les fondations d'un SI doivent être solides afin de supporter l'ensemble des fonctionnalités qui différencient une entreprise. Faire appel à des experts pour colmater des brèches ponctuellement est une stratégie à court terme, qui mènera à coup sûr dans l'impasse. Il faut voir plus 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.

Genèse du mouvement

La première inspiration du mouvement vient certainement du livre *The Pragmatic Programmer* [PRAGMATIC]. Le terme est utilisé pour la première fois par Pete McBreen [MCBREEN].
Le mouvement prit réellement de l'ampleur lorsque Robert C. « Uncle Bob » Martin proposa, en vain, d'ajouter une cinquième valeur au manifeste agile : *Craftsmanship over Execution.*

Cet échec a mené, de concert entre plusieurs membres fondateurs du manifeste agile, à la rédaction du Software *Craftsmanship Manifesto*. Publié en 2009, il se veut une extension et un complément nécessaire au manifeste agile.

Initier

Les technologies, les outils, les frameworks ainsi que les besoins des utilisateurs changent continuellement.

Il s'agit d'une caractéristique aujourd'hui reconnue du développement logiciel. La durée de vie d'un produit est donc directement liée à sa capacité à évoluer. C'est pour cela que les craftsmen (ou craftswomen) cherchent constamment à construire des logiciels de qualité, bien conçus et aptes à évoluer.

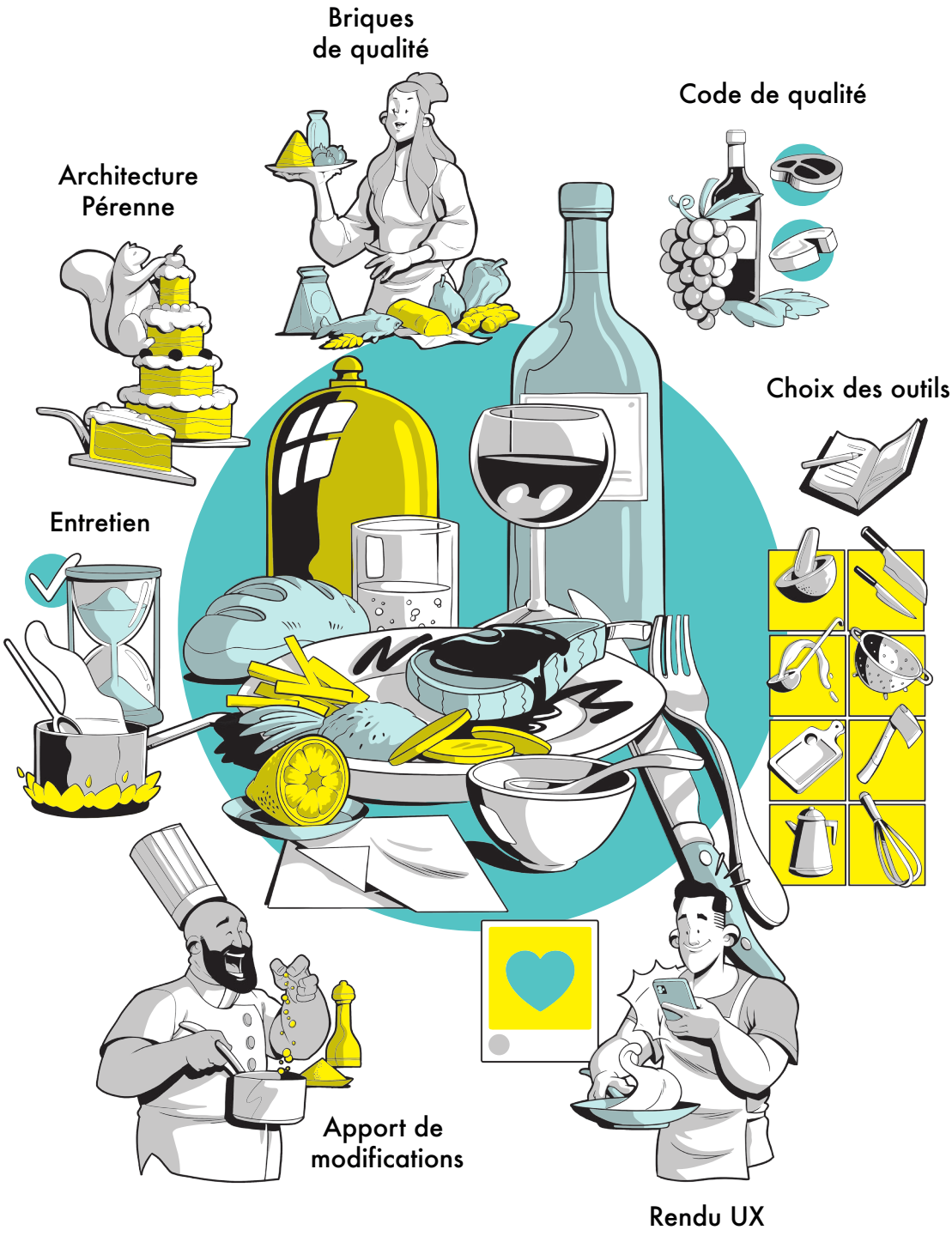
De plus, la passion, le partage et l'apprentissage continu sont les ingrédients nécessaires à toute innovation. Il n'est donc pas étonnant de constater que la plupart des technologies et outils innovants sont le fruit du travail d'équipes passionnées.

Définissez les bases de la construction

La qualité d'un logiciel repose sur :

- **la définition** d'une architecture pérenne et modulaire ;
- **la qualité** des briques logicielles utilisées ;
- **la qualité intrinsèque** du code qui permet de les assembler ;
- **le choix** des outils et le rendement qu'ils procurent aux développeurs ;
- **le rendu final** aux yeux des utilisateurs ;
- **la facilité** à apporter des modifications à travers le temps, 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, les objets connectés ou la vente en ligne - serai-je en mesure de faire évoluer mon SI sans tout revoir de fond en comble ? "



Les bases de la qualité logicielle

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 quant à lui une vision plus opérationnelle : le projet sera considéré comme un succès si l'utilisateur final est satisfait 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 fiable, évolutif et pourra « lui survivre ».

En prônant une qualité sans compromis, teintée d'un grand pragmatisme, le mouvement Software Craftsmanship donne les moyens d'atteindre ces objectifs. En poussant les craftsmen du développement logiciel à améliorer de manière quotidienne leur production et 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.

Constituez votre équipe craft

La réussite d'un projet complexe repose avant tout sur les humains. Le développement logiciel ne fait pas exception.

Si se reposer sur un « super-héros » peut fonctionner ponctuellement, c'est une approche qui n'est pas tenable dans la durée. C'est sur une équipe, au sein de laquelle chacun apporte son savoir-faire, ses méthodes, ses outils et ses idées, qu'il faut compter.

Trouver les bonnes personnes est crucial pour créer un environnement qui permettra à chacun de s'exprimer à sa juste valeur. Une équipe soudée se construit autour de plusieurs 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 et la production d'un code toujours opérationnel même après plusieurs années sont des fiertés de tous les jours. Un craftsman est enclin à s'impliquer et s'investir en améliorant 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. Comme dans toutes les disciplines : ceux qui excellent sont ceux qui pratiquent avec passion.

L'implication

Produire des lignes de code n'est pas une finalité en soi. Une 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. Il s'agit, au contraire, d'une force : un développeur sera toujours plus efficace s'il comprend les problématiques en amont et en aval, qu'elles soient fonctionnelles ou techniques.

Les équipes de développement ne sont pas constituées de simples exécutants. La qualité de la modélisation technique des projets n'est pas un asset mineur pour le SI, elle est primordiale. La remise en cause de choix architecturaux ou fonctionnels ne sont pas des affronts faits aux penseurs mais plutôt le reflet d'une volonté commune d'atteindre l'excellence. L'équipe devient alors force de proposition.

Le pragmatisme et le bon sens

Un bon craftsman ne perd jamais de vue le but de son travail : produire un logiciel de qualité, certes, mais qui répond avant tout aux besoins de ses utilisateurs. Il est parfois tentant d'expérimenter la dernière technologie dont tout le monde parle, mais c'est le contexte qui doit guider ces choix. Quels sont les bénéfices de chaque solution ? Sont-ils à la hauteur du coût et des risques qu'elles impliquent ? Les meilleurs craftsmen 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.

Cette légitimité peut reposer sur les réalisations passées, la réputation, la recommandation par des personnes de confiance, etc.

Mais plus généralement, elle se construit tout au long de la collaboration. 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 métier. Démontrer qu'une fonctionnalité repose sur du code de qualité, parfois plus long à produire qu'un correctif sur un coin de table, n'est pas une mince affaire. C'est cette confiance mutuelle qui permettra de remplacer le conflit d'opinions par des discussions constructives.

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 amène 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 et pourra même pénaliser l'équipe.

" Un craftsman est enclin à s'impliquer et s'investir en améliorant le code qu'il produit, en cherchant toujours la solution la plus adaptée 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. "

Créez une équipe unifiée et complémentaire

Ça peut sembler contre-intuitif, mais se reposer sur un « super-héros », qui possède la majorité de la compétence technique ou la plus large connaissance fonctionnelle, est dangereux. 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 efficace pour résoudre ce problème est de constituer des équipes pluridisciplinaires et techniquement homogènes. N'obligez pas un développeur à avoir un domaine réservé dont il sera le seul spécialiste. Favorisez plutôt un environnement où tous les membres de l'équipe sont complémentaires mais pas indispensables.

Il est bien évidemment difficile d'atteindre un tel but, des disparités existent forcément. Chercher à les lisser en tirant l'équipe vers le haut est l'un des principaux objectifs poursuivis par le Software Craftsmanship.

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 plus simple reste bien sûr d'avoir une équipe colocalisée.

Cependant, l'existence d'équipes distribuées (télétravail, offshoring, etc.) est désormais une réalité. Dans ce cas, il est nécessaire de se doter d'outils appropriés (vidéoconférence, partage d'écran, tableau blanc virtuel, outils de développement collaboratif, etc.) mais également de bonnes pratiques qui permettent de minimiser les impacts de l'éloignement.

Favoriser l'auto-organisation des équipes, mettre en place une démarche d'amélioration continue, visualiser le flux des travaux en cours, la dette technique et les problèmes à résoudre sont autant de pratiques nécessaires pour atteindre notre objectif. Les outils agiles peuvent vous y aider.

Vient ensuite la formation en interne. Le partage des connaissances dans l'équipe permet notamment aux membres les plus débutants de progresser rapidement. Cette évolution passe par l'observation et surtout la pratique. Dans le monde du développement logiciel, l'utilisation des pratiques de l'eXtreme Programming (XP) donne des résultats convaincants.

La revue de code

L'équipe doit avoir recours à la revue de code : un ou plusieurs équipiers relisent une partie du code avec son auteur, émettent des critiques constructives permettant à l'auteur de retravailler le code en conséquence. Des revues fréquentes favorisent la mise à niveau du code et évitent de laisser des zones inexplorées devenir une dangereuse gangrène pour l'ensemble du logiciel. Elles renforcent également la notion de « responsabilité collective » de la base de code.

Les revues de code peuvent être un frein à la fluidité dans la mise à disposition des modifications. D'autres pratiques de partage, complémentaires, peuvent être mises en œuvre en amont pour éviter cet écueil.

Le pair programming

Le pair programming favorise grandement le partage de la connaissance au sein de l'équipe par l'utilisation d'un seul poste de travail pour deux personnes. Cette mutualisation de deux cerveaux sur une même tâche peut apparaître comme une perte sèche et un gaspillage de temps. 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 vue distincts. De ce brainstorming émerge généralement une solution de plus haut niveau.

La communication permanente entre les deux membres du binôme a aussi des effets indirects encore plus intéressants :

- C'est la meilleure façon pour chacun d'apprendre de l'autre : il est fréquent, lors de sessions de pair programming, d'entendre des échanges sur les raccourcis utilisés, les raisons d'une séparation en méthodes, etc.
- 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.

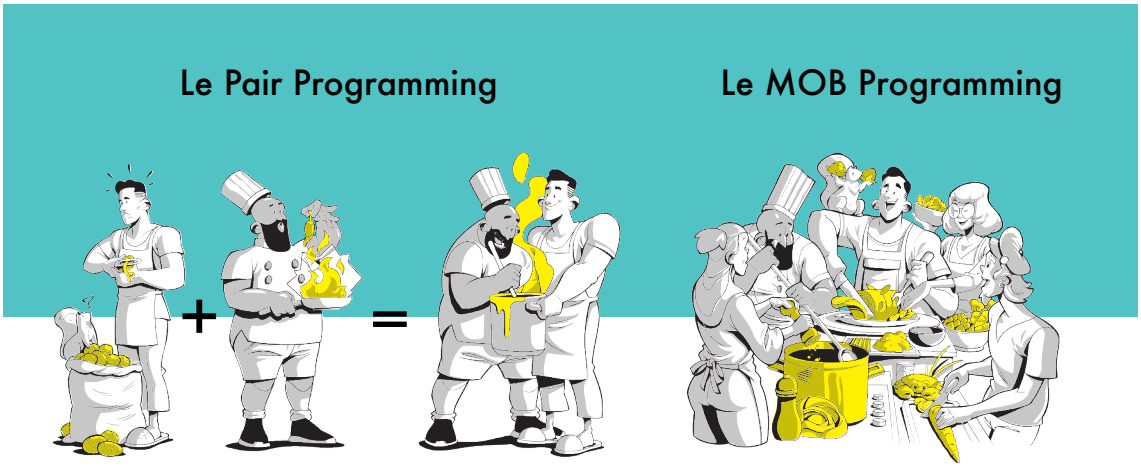
Le mob programming

Avec le mob programming, pratique plus récente, toute l'équipe travaille ensemble sur le même sujet, en même temps et sur la même machine. Cette méthode de travail inclut évidemment les développeurs mais également les autres profils tels que les testeurs, les designers et toute personne faisant partie de l'équipe de réalisation.

Cette technique exacerbe certains bénéfices du pair programming : meilleure appropriation collective du code, diminution du besoin en revue de code, augmentation de la qualité, réduction des risques liés aux absences et aux départs, facilitation de l'intégration de nouveaux membres.

Elle favorise la disparition des silos entre les différents profils d'une équipe et diminue le besoin de synchronisation.

De plus, la pratique régulière du mob programming est le meilleur moyen pour les développeurs d'améliorer leurs pratiques à la fois individuelles et collectives.



Pair et Mob Programming

De la pluridisciplinarité

Avec le mob programming, pratique plus récente, Grouper au sein d'une même équipe l'ensemble des compétences nécessaires à la création et la maintenance d'un produit offre beaucoup d'avantages par rapport à une approche plus « silotée » de l'organisation. Avec une telle équipe, il est plus facile de créer une cohésion de groupe, une entraide pour avancer vers un but commun : concevoir et créer les prochaines versions du produit. Plutôt que de rejeter la faute sur les UI/Architectes/Ops/Analystes (rayez les mentions inutiles), une équipe soudée et pluridisciplinaire se portera mutuellement assistance pour lever les obstacles rencontrés. La communication est plus facile au sein d'une même équipe et les échanges deviennent plus facilement constructifs. Autonomes et responsables (You Build It, You Run It), les équipes pluridisciplinaires faciliteront aussi la montée en compétence de chacun sur les connaissances et techniques des différents métiers qui la composent. Il en résulte une résilience accrue de l'équipe face à la perte temporaire ou permanente d'un de ses membres.

IDENTIFIER UN CRAFTSMAN

Ces questions peuvent vous aider à identifier un craftsman :

- Considère-t-il le développement logiciel comme son métier ou uniquement une étape dans sa carrière avant de passer à un autre rôle ?
- Comprend-il qu'il est difficile de produire du code simple ?
- Possède-t-il une culture d'apprentissage continue ? Par quels moyens ? Quelles sont ses lectures, les événements auxquels il assiste ? Qu'a-t-il appris de nouveau dernièrem—ent ?
- Possède-t-il une culture du partage ? Comment partage-t-il ses connaissances avec les autres ?
- Préfère-t-il travailler seul ou en équipe ? Est-ce qu'il connaît les avantages du pair programming ? Est-ce qu'il comprend l'importance du feedback ?
- Que pense-t-il des tests ? Connaît-il différents types de tests ? Quelles pratiques de tests connaît-il ?
- Connaît-il l'importance de la collaboration avec le client ? Comment cette collaboration se traduit-elle dans son travail et les pratiques qu'il utilise ?

Take away

Initier

Définir

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 : stabilité, évolutivité, lisibilité.

Constituer et entretenir

Constituer et entretenir des équipes de craftsmen passionnés, impliqués, pragmatiques et humbles.

Favoriser

Favoriser la communication et les pratiques de collaboration (pair programming, etc.) afin de maintenir l'homogénéité de l'équipe et de la qualité du logiciel.

Construire

Une fois les bases posées, il reste à bâtir.

C'est un travail de longue haleine, qui demande une attention et une remise en question quotidiennes.

Le mouvement Craftsmanship amène un ensemble d'outils et de pratiques qui doivent aider l'équipe au quotidien, mais surtout éviter les dérives dans le temps.

Choisissez les bons outils

Quels outils un craftsman utilise-t-il ? Ceux dont il dispose. Avec lesquels est-il le plus efficace ? Avec ceux qu'il maîtrise.

Dans le domaine du développement logiciel, les « outils » incluent les logiciels que le développeur manipule pour mener à bien sa mission. Quelques exemples :

- **les environnements** de développement intégrés (IDE) ;
- **les langages de programmation** ;
- les outils de **test et de mock** ;
- **les frameworks** et bibliothèques réutilisables ;
- **les outils de gestion** de code source (*eg.* git) ;
- **les outils** de build (*eg.* Maven).

Bien sûr, il est indispensable que chaque développeur prenne le temps de comprendre comment fonctionnent les outils qu'il emploie et d'apprendre comment les exploiter au maximum de leurs capacités. En mettant à disposition des développeurs les outils qu'ils maîtrisent, ils pourront ainsi se concentrer sur le plus important : créer des applications qui répondent de la meilleure manière aux besoins des utilisateurs.

Faut-il payer une licence à un développeur qui le demande pour mieux travailler ? La réponse est généralement oui. Pour ces raisons, payer une licence afin de bénéficier d'un outil adapté pour mieux répondre *in fine* aux besoins métiers, s'avère en règle générale un investissement très rentable.

Dans le même ordre d'idées, développer un framework « maison » et exiger des développeurs qu'ils l'utilisent lorsque des solutions éprouvées et activement maintenues existent déjà, s'avère généralement contre-productif. Ne vous rendez pas victime du syndrome « Not invented here ».

Au-delà des outils que nous venons d'aborder, un craftsman s'intéresse à ce qui concerne son quotidien hors de son domaine d'expertise. Qu'il s'agisse des principes agiles et leurs mises en œuvre ou de la culture DevOps et des pratiques associées, le craftsman agrandit la sphère de ses compétences et garde en tête la volonté de s'améliorer pour mieux servir ses clients et leurs utilisateurs.

> " 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. "

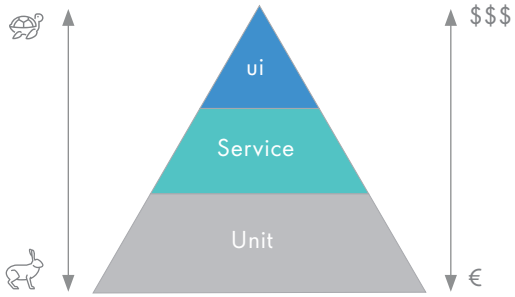
Obtenez du feedback

« Mon code répond-il au besoin des utilisateurs ? Est-il suffisamment clair ? Est-il facilement maintenable et évolutif ? Ai-je introduit des bugs ? » Autant de questions que tout développeur se pose pendant son travail au quotidien. Un craftsman va avant tout chercher un feedback rapide afin d'ajuster la trajectoire et livrer un logiciel de meilleure qualité.

Plusieurs pratiques vont nous permettre de raccourcir la boucle de feedback et d'avoir des retours en permanence. Parmi elles, on retrouve les méthodes de travail en groupe telles que le pair / mob programming et les revues de code déjà citées qui permettent d'avoir un retour en temps réel de la part de ses collègues. On retrouve aussi des pratiques de test et d'intégration continue, capables de guider les développeurs et de fournir des informations sur l'état du code en continu.

Les tests

Qu'ils soient manuels ou automatiques, les tests sont un élément de feedback essentiel. Automatisés, ils permettent de gagner du temps, de réduire le risque d'erreur humaine, de rejouer les tests autant de fois que nécessaire et à tout moment.



Pyramide de Mike Cohn

La pyramide de Mike Cohn représente, de façon simplifiée, les types de tests selon leur granularité. Plus la granularité est fine, plus les temps d'exécution sont courts et moins les coûts de maintenance sont importants.

Le craftsman va chercher à équilibrer ces tests en ayant le modèle de la pyramide en tête afin de maximiser le feedback et minimiser l'effort de mise en place et de maintenance des tests.

Les tests de faible granularité fournissent un filet de sécurité qui permet d'envisager sereinement la suite des développements : si chaque méthode réagit individuellement comme je l'ai spécifié, alors la combinaison de toutes les méthodes de mon code devrait continuer à fonctionner sans régressions. Les anomalies introduites seront détectées plus rapidement et leur coût de correction grandement diminué.

En complément, des tests de forte granularité permettent de valider que le comportement global du logiciel est celui attendu, en se concentrant sur les principaux scénarios d'utilisation.

En parallèle, la refactorisation, 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.

Par ailleurs, les tests peuvent être considérés comme une « documentation vivante » du code, en particulier lorsqu'ils intègrent les principes de spécification par l'exemple [GOJKO].

Ils peuvent aussi guider la conception : la difficulté de tester du code est souvent le symptôme d'une mauvaise conception et le signe qu'il est nécessaire de le reprendre en main.

Test-Driven Development

Coder les tests en premier favorise un design émergent piloté avant tout par la réponse à un besoin donné plutôt que par les contraintes techniques des outils employés : les développeurs apportent d'abord une réponse au besoin et y intègrent ensuite les contraintes techniques qui apparaissent naturellement. Cela permet d'avoir un design plus simple, un code compréhensible et favorise l'émergence d'une architecture pertinente répondant au besoin donné uniquement.

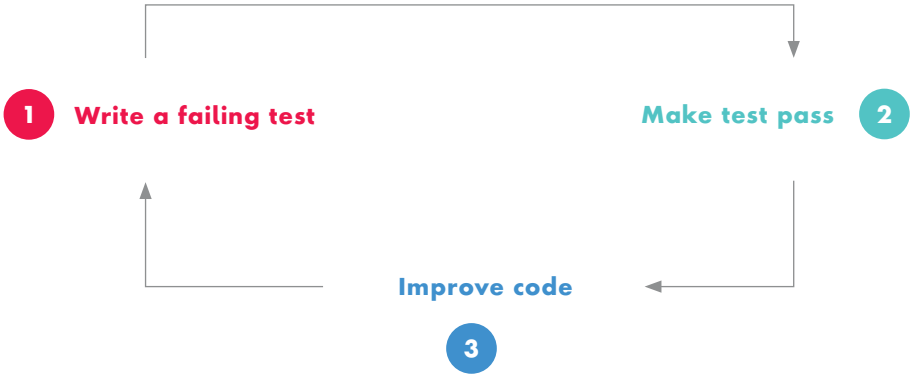
Les équipes de craftsmen ont systématisé cette approche, à travers la pratique du Test-Driven Development (TDD) : l'ajout de chaque nouvelle fonctionnalité est piloté par les tests, en suivant de façon itérative le principe Red / Green / Refactor :

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.

Le cycle est ensuite répété en écrivant un nouveau test à chaque fois pour avancer de façon itérative et incrémentale. La taille des étapes doit toujours être petite. Si le nouveau code ne permet pas rapidement de faire passer le nouveau test, ou si d'autres tests échouent de manière inattendue, le développeur doit annuler ces modifications et revenir au dernier état stable.



Les 3 étapes du TDD

Les avantages les plus importants du TDD sont :

- construire une boucle de feedback très courte ;
- constituer un harnais de test de non-régression ;
- inciter à l'écriture de code simple ;
- favoriser la testabilité du code ;
- inciter à l'adoption d'une une approche « Contract First » des tests ;
- maintenir la qualité en continu.

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'intégration continue

L'intégration du code des membres de l'équipe plusieurs fois par jour permet d'obtenir un feedback rapide et régulier sur l'état de la compilation, des tests automatisés ou l'exécution d'outils d'analyse de la qualité. C'est une pratique qui est aujourd'hui indispensable pour détecter les régressions au plus tôt et 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 DevOps), 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)

La collaboration entre des profils complémentaires pour construire le comportement attendu de l'application est au cœur du Behavior-Driven Development. Canoniquement, 3 profils, appelés les 3 Amigos, sont représentés dans les discussions : l'expertise métier, le développement et le test. Dans la pratique, il arrive que 2 profils soient cumulés par une même personne.

Qu'attendre de cette étroite collaboration ? Plusieurs bénéfices :

- une meilleure compréhension du métier par l'équipe de développement ;
- la création et le renforcement d'un langage commun entre l'équipe de développement et l'expert métier ;
- des spécifications par l'exemple du comportement qui devra être implémenté dans l'application.

Pour faciliter la collaboration et composer avec l'agenda souvent surchargé de l'expert métier, certains types d'ateliers peuvent être très utiles. Parmi eux, l'Example Mapping propose, en 25 minutes, aux 3 Amigos d'explorer une User Story, les règles métier qui la composent et d'illustrer ces règles par des exemples. Une fois le temps écoulé, les participants sauront décider si la User Story est prête pour être développée (selon leur Definition of Ready).

Cerise sur le gâteau du Behavior-Driven Development, les exemples produits par les 3 Amigos peuvent être exploités pour écrire les tests métier validant le bon fonctionnement et la non-régression de l'application. Ces tests constituent alors la base d'une documentation fonctionnelle toujours à jour de l'application : une documentation vivante.

Le Domain-Driven Design (DDD)

Toujours dans cette idée de collaboration avec le métier, le DDD est une approche de conception qui a pour objectif d'aligner le logiciel avec le métier qu'il représente, par un code expressif et une architecture flexible.

Une entreprise a bien souvent plusieurs métiers. Facturation, livraison, etc. Ils sont identifiés comme des « domaines » avec des besoins différents. Chacun d'eux est modélisé dans un contexte aux limites bien définies. Pour éliminer les biais d'incompréhension, cette modélisation prend pour base la construction d'un langage commun (« Ubiquitous Language »), partagé aussi bien par les développeurs que les experts métiers, dans le code comme dans les discussions.

Cette volonté générale d'alignement avec le métier facilite les échanges et améliore la qualité du logiciel, le rendant plus apte à évoluer. Elle s'accompagne d'outils qui vont de patterns de conception à des principes d'architecture de haut niveau. Les maîtriser est un atout inestimable pour tout développeur.

Le DDD comme le BDD vise à casser le mur entre penseurs et faiseurs. Pour bien faire son métier, le craftsman a besoin d'une bonne compréhension du métier et il doit donc prendre part activement à la conception.

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.

Le Collective Code Ownership doit devenir le mot d'ordre des équipes. Quand bien même chaque développeur présente ses spécificités, il est important que l'équipe s'accorde sur des pratiques homogènes, notamment en termes de configuration d'outil ou de style de code, sans quoi la collaboration s'en trouvera compliquée, notamment par des débats stériles.

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 atout maîtrisé par l'ensemble de l'équipe. Ils pourront donc être autonomes et réactifs sur l'ensemble du logiciel, même en l'absence du référent en la matière.

La propriété partagée ne s'arrête pas au code. Elle est, par exemple, aussi valable en production : les réussites sont fêtées en équipe au même titre que le traitement des incidents est la responsabilité de l'équipe.

Take away

Construire

Laisser

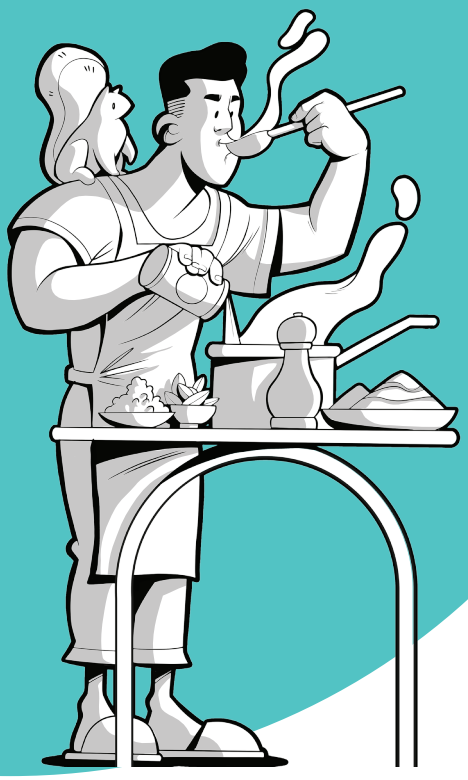
Laisser les craftsmen choisir leurs outils, dans la mesure du possible et dans le respect des contraintes de l'entreprise.

Encourager

Encourager un environnement propice à la réalisation de tests, au maintien et au nettoyage (refactoring) fréquent du code en formant les équipiers sur ces thématiques.

Favoriser

Favoriser la collaboration avec le métier en construisant un langage commun.



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, il est nécessaire de disposer d'une base de code propre et bien entretenue. Cela implique également de placer l'innovation au niveau des méthodes et outils utilisés par vos équipes.

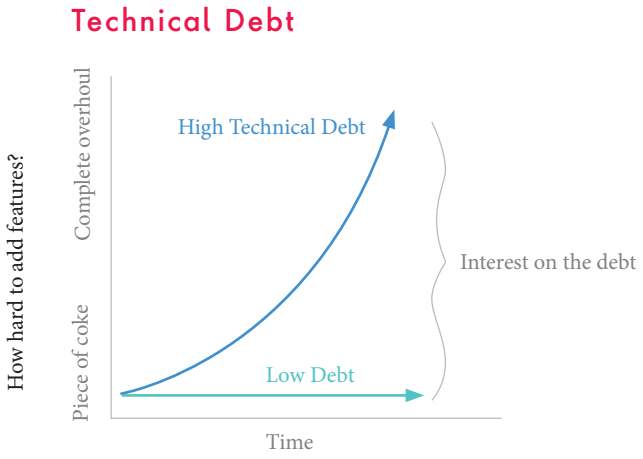
Aujourd'hui, les Software Craftsmen ont développé des réseaux et des outils leur permettant d'être toujours dans les sphères d'excellence.

Conservez une base saine

Parfois, il est pratique de prendre temporairement des libertés avec le niveau de qualité attendu. Qu'il s'agisse de corriger au plus vite une anomalie ou d'expérimenter une fonctionnalité en situation réelle, il est indispensable de faire comprendre à toutes les parties prenantes que les altérations de la qualité du produit sont volontaires et surtout temporaires. Il est nécessaire de visualiser clairement où et quand le produit a été volontairement abîmé, dans le but de rétablir une situation saine et maintenable une fois l'urgence passée. Ce concept d'altération délibérée et temporaire de la qualité du produit porte le nom de dette technique.

« Un crédit vous engage et doit être remboursé. Vérifiez vos capacités de remboursement avant de vous engager. », peut-on lire dans les communications associées au crédit. Contracter une dette technique devrait être fait en ayant cette mention à l'esprit pour éviter de diriger le produit vers une situation irrattrapable où l'ajout de fonctionnalités coûte toujours plus cher et les anomalies s'accumulent plus vite que l'équipe n'arrive à les corriger.

« Usine à gaz », « bloatware », « tas de boue » : les qualificatifs ne manquent pas pour décrire un produit surchargé de fonctionnalités et aux besoins de ressources matérielles exagérés par rapport au service rendu. Ajouter des fonctionnalités sans jamais en retirer rend toujours plus laborieuse et coûteuse la maintenance du produit. En retirant les fonctionnalités inutilisées ou trop peu utilisées, on favorise la maintenabilité du produit, limite l'impact du départ des « sachants », renforce la pertinence du produit et maîtrise les coûts d'exploitation. Aussi, dans une optique de réduction de notre empreinte environnementale, c'est un très bon moyen pour permettre à nos utilisateurs de prolonger la durée de vie de leurs terminaux, dont la fabrication est le plus grand facteur d'émission de gaz à effet de serre de l'écosystème du numérique[2].



2- Étude « Empreinte environnementale du numérique mondial » https://www.greenit.fr/2019/10/22/12982/

Faites évoluer

La vie d'une application s'étale sur plusieurs années. Pour espérer rester pertinent, un produit doit pouvoir évoluer avec son marché et le paysage dans lequel il prend place : des usages apparaissent tandis que d'autres disparaissent, les technologies choisies à une époque ne sont plus forcément adaptées par la suite, la loi et les réglementations changent, etc.

Sur le plan technologique, plusieurs pratiques peuvent servir l'ambition de pertinence et de pérennité d'un produit. Par exemple, retarder les choix technologiques majeurs jusqu'au dernier moment raisonnable permet d'effectuer des choix plus avisés qu'au début de la vie du produit. Ainsi, le risque de mal choisir est grandement réduit, tout comme celui de contracter une dette technique inutile.

Certaines architectures logicielles, comme l'Architecture Hexagonale[3], facilitent le remplacement de briques technologiques dans les contextes où le métier est moins trivial qu'un CRUD (Create/Read/Update/Delete).

Garder à jour les dépendances d'une application au fil de l'eau permet de bénéficier des derniers correctifs, profiter des évolutions et faciliter les migrations. Des outils comme Renovate ou Dependabot proposent même de faciliter cette tâche en automatisant tout ou partie de ce processus. Bien employés et intégrés dans votre système de revue de code, ils deviennent presque comme un équipier spécialisé.

" 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. "

3- https://blog.engineering.publicissapient.fr/2016/03/16/perennisez-votre-metier-avec-larchitecture-hexagonale (QR code accessible dans la bibliographie)

Partagez la connaissance en interne

Nous avons précédemment vu qu'un premier niveau de partage peut être atteint grâce aux pratiques d'eXtreme Programming. Il reste cependant souvent limité à quelques individus. Afin de garantir la cohésion de l'équipe au sens global (y compris les développeurs non présents physiquement), il est préférable de favoriser des échanges réguliers (tous les trimestres, par exemple) mais aussi diversifiés. L'investissement paraît important, mais la cohésion et la qualité globale de l'équipe en seront renforcées.

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 sont inefficaces : proposées en fin de soirée, avec des ordres du jour bricolés 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 de la connaissance est un vrai plus pour vous, mettez-y les moyens. Chez Publicis Sapient France, nous avons instauré une journée mensuelle de partage de la connaissance. 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 vidéoprojecteur par exemple), mettez-le à disposition de vos équipes de développement. De nombreuses pratiques issues du mouvement Software 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 d'acquérir des connaissances théoriques mais aussi de profiter des retours de terrain.

Ce format est facilement adaptable pour des présentations à distance grâce aux outils de vidéoconférence.

• 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'aguerrit et devient « 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 Dojos sont un moyen pour contourner ce problème : plusieurs personnes s'entraînent, en binôme ou en groupe, pour résoudre un même problème, apprendre ou perfectionner une pratique, un outil ou une méthode 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.

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 émergent en particulier) ou personnel (mise à niveau technologique, découverte d'un nouveau framework, etc.).

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.

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

Échangez avec la communauté

Les User Groups et Meetups

Ces dernières décennies ont vu 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 React, 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 se fera l'ambassadeur de cet apprentissage au sein de l'équipe dès le lendemain.

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, NCraft). 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, mais la prise en charge financière par l'entreprise est un investissement sur la connaissance qui est souvent payant pour l'entreprise.

7- https://docs.docker.com/docker-cloud/builds/image-scan/ (https://dockr.ly/2QCvo6o)

Take away

Entretenir

Identifier

Identifier et visualiser la dette technique pour pouvoir la maîtriser.

Choisir

Choisir une architecture évolutive et retarder les choix technologiques pour une meilleure pertinence et pérennité du produit.

Aménager

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

Miser sur la communauté pour ne pas rester en marge des innovations.

Conclusion

Il y a 10 ans, Marc Andreessen écrivait dans le Wall Street Journal « Why Software Is Eating the World ». Les événements de 2020 ne permettent plus d'en douter : la majorité des entreprises doivent devenir des entreprises de développement logiciel, c'est une question de survie.

Un des 12 principes sous-jacents du Manifeste Agile prône une attention continue à l'excellence technique et à une bonne conception. C'est sans doute la clé : cultiver un terreau fertile à l'épanouissement du Software Craftsmanship est un levier dans toute entreprise, quelle que soit sa taille.

Le développement n'est pas une phase dans la carrière d'un ingénieur, mais bien un métier à part entière, dans lequel les personnes s'investissent avec passion mais aussi et surtout avec responsabilité et la volonté de bien faire.

Embrasser le mouvement Software Craftsmanship vous permettra de construire et de garder à flot un système d'information de qualité, capable de rendre actionnable la créativité et la réactivité de vos équipes métiers.

Et même si vous ne pourrez peut-être pas attirer les meilleurs spécialistes dans tous les domaines, nous vivons à une époque où le savoir est accessible et son partage une vraie volonté. Cette culture du partage donne la possibilité à chacun de progresser et d'en faire profiter le groupe dans son ensemble.

Take away

Craftsmanship

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 : stabilité, évolutivité, lisibilité.

- Constituer et entretenir des équipes de craftsmen passionnés, impliqués, pragmatiques et humbles.

- Favoriser la communication et les pratiques de collaboration (pair programming, etc.) afin de maintenir l'homogénéité de l'équipe et de la qualité du logiciel.

Construire

- Laisser les craftsmen 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 en formant les équipiers sur ces thématiques.

- Favoriser la collaboration avec le métier en construisant un langage commun.

Entretenir

- Identifier et visualiser la dette technique pour pouvoir la maîtriser.

- Choisir une architecture évolutive et retarder les choix technologiques pour une meilleure pertinence et pérennité du produit.

- 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

Livres

Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 2008

[PRAGMATIC] Andrew Hunt et David Thomas, *The Pragmatic Programmer: From Journeyman to Master*, 1999

Kent Beck et Cynthia Andres, *Extreme Programming Explained: Embrace Change* - 2nd Edition, 2004

[MANCUSO] Sandro Mancuso, *The Software Craftsman: Professionalism Pragmatism*, Pride, 2014

[MCBREEN] Pete McBreen, *Software Craftsmanship, The new Imperative,* 2001

Eric Evans, *Domain-Driven Design, Tackling complexity in the Heart of Software*, 2003

Gáspár Nagy, Seb Rose, The BDD Books - Discovery, 2018

[GOJKO] Gojko Adzic, *Specification by Example*, 2011

Sur le Web

[MANIFESTO] https://manifesto.softwarecraftsmanship.org/

Pour aller plus loin

Articles Expertise Craft

Strong Style Pairing (vidéo)

À propos de la dette technique

DDD - La conception qui lie le fonctionnel et le code

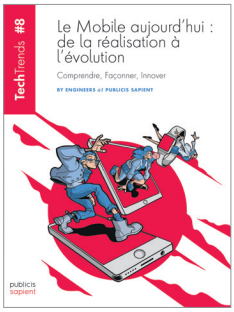
Fiches de lecture du livre de Sandro Mancuso

Pérennisez votre métier avec l'architecture hexagonale !

À lire et à relire

Pour télécharger l'un des TechTrends en version éléctronique (pdf), rendez-vous sur le site https://blog.engineering.publicissapient.fr/publications/

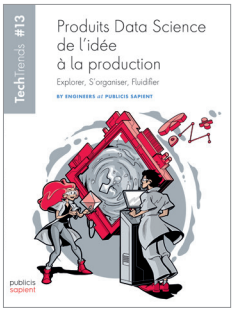
Pour en savoir plus sur le Data, le Cloud, le Web, les architectures Java, la mobilité et l'agilité, rendez-vous sur blog.engineering.publicissapient.fr



Techtrends #8
Mobile
Comprendre, Façonner, Innover

Techtrends #9
Cloud
Préparer sa migration, Séléctionner son offre, Choisir ses technologies

Techtrends #10
Studio
Démarrer son projet, Piloter son projet, Pérenniser son produit



Techtrends #13
Produits Data Science
Explorer, S'organiser, Fluidifier

Techtrends

#12 Conteneurs
#11 Internet of Things
#7 Back
#6 DataLab
#5 Front
#3 Agilité
#2 DevOps
#1 Data

Autres parutions

Scrum Master Academy
Le Guide du Scrum Master d'élite

Les Communautés de Pratique en Pratique
Le Mini Guide

Product Academy
Le guide des Product Managers et des Product Owners d'élite

Les auteurs

Anis Chaabani

Sylvain decout

Édouard Siha

Auteurs de la 1ère édition

Anne Beauchart, Bastien Bonnet, Xavier Bucchiotty, Alexandre Dergham, Mathieu Dulac, Benoit Guerout, Christophe Heubès, Sébastian Le Merdy, Diego Lemos, Pablo Lopez, Christophe Pelé,

Design graphique : **micheletmichel.com** / Illustrations: **lasaintepaire.com**

Publicis Sapient
94 Avenue Gambetta, 75020 Paris
+33 (0)1 85 56 91 79
engineering@publicissapient.com

Toutes les informations sur :
publicissapient.fr