The Software craftsmanship

Table des matières

Auteur : Aziz MOHAMED BEN ALI

Introduction

De manière générale, les entreprises font un focus sur la réduction des coûts dans le cadre de la réalisation d'un produit répondant à leurs besoins. Elles veulent un produit qui soit disponible le plus rapidement possible avec des coûts réduits. Cela peut paraître un choix judicieux pour une entreprise dans le cadre de l'optimisation des budgets, mais cela n'est pas une bonne solution pour un projet lorsque cette rapidité est obtenue en sacrifiant les bonnes pratiques et la maturité technique, car deux problèmes apparaissent.

Un produit mal conçu

A force de vouloir faire les choses rapidement pour sortir un produit le plus tôt possible dans le but de réduire les coûts de développement et de rentabiliser le produit le plus rapidement possible on va se retrouver avec un produit mal conçu. En effet, la stratégie du « vite fait et peu coûteux » n'est pas une stratégie gagnante sur le long terme. Les conséquences de ce choix sont lourdes. En effet, avec le temps les technologies évoluent et faire suivre ces évolutions technologiques sur un produit mal conçu sera de plus en plus difficile à cause d'une mauvaise architecture logicielle. La maintenance sera elle aussi de plus en plus complexe à cause d'un manque d'organisation et de cohérence dans le code. Les correctifs sur les nombreux bugs constatés en production seront aussi de plus en plus difficiles à apporter à cause d'un manque de clarté du code.

Toutes ces difficultés vont, sur le long terme entraîner une explosion des coûts et seront préjudiciables à l'entreprise dans le sens où la maintenabilité sera très complexe et longue à tel point que le projet peut être abandonné pour être entièrement redéveloppé (from scratch).

Développeur, un rôle essentiel mais souvent mal compris

Quand une entreprise veut faire les choses rapidement pour mettre son produit en production ou bien parce qu'il faut respecter des délais serrés (voire très serrés), il y a des sacrifices à faire et le rôle du développeur en pâtit. Le développeur est vu comme un simple exécutant. Il est présent pour écrire du code. Le client a un besoin que le développeur traduit en langage informatique. Son rôle est réduit au strict minimum et fait du développeur un élément de second rôle avec peu de responsabilité alors que c'est un élément de premier plan.

Auteur : Aziz MOHAMED BEN ALI

C'est à partir de ces deux problématiques récurrentes du monde de l'industrie logicielle que le software craftsmanship qui a émergé dans les années 2000 (avec un manifeste en 2009) apporte des réponses. Le software craftsmanship (artisanat du logiciel), est un concept, une philosophie qui accorde une importance primordiale à la conception d'un produit et au rôle du développeur. En effet, cette philosophie apporte des principes, des valeurs, un état d'esprit permettant de faire d'un produit un produit bien conçu et de faire du développeur un élément de premier plan.

Le manifeste du Software Craftsmanship

Voici ce que contient le manifeste craft (https://manifesto.softwarecraftsmanship.org/) :

- Pas seulement des logiciels opérationnels, **mais aussi des logiciels bien conçus**.
- Pas seulement l'adaptation aux changements, **mais aussi l'ajout constant de la valeur.**
- Pas seulement les individus et leurs interactions, **mais aussi une communauté de professionnels.**
- Pas seulement la collaboration avec les clients, **mais aussi des partenariats productifs.**

Le manifeste craft est complémentaire au manifeste agile (ils sont étroitement liés) dans le sens où ce dernier se concentre sur l'organisation de l'équipe et la collaboration avec le client, alors que le manifeste craft met l'accent sur la qualité technique et le professionnalisme/responsabilité du développeur.

Afin d'illustrer les 4 principes de ce manifeste nous allons ci-dessous voir quelques pratiques permettant de mettre l'accent sur la qualité technique et le professionnalisme du développeur afin de résoudre les problématiques mentionnées au début de cet article mais aussi ce qu'est un développeur dans le monde du craft.

Les pratiques

- Mob programming :

  Le **Mob Programming** est une pratique presque similaire au **Pair Programming**, mais impliquant l'ensemble de l'équipe (au moins trois personnes). Tous travaillent ensemble sur une même user story, devant un seul poste. Une personne est au clavier et rédige le code, tandis que les autres guident et proposent les solutions. Toutes les 10 minutes (maximum 15 minutes), les rôles changent, permettant à chacun de coder à tour de rôle.

  Cette rotation favorise la montée en compétence de l'équipe : chaque membre apprend des autres, qu'il s'agisse des outils, des techniques ou de la connaissance métier. L'objectif est de renforcer la qualité du produit tout en développant la cohésion et l'expertise de l'équipe ainsi que sa maturité.

- Le Test-Driven-Development :

  Le **TDD** est une méthodologie itérative qui permet d'améliorer la qualité de code via la conception, l'implémentation et la réduction d'anomalies via un développement guidé par les tests. En effet, cette méthodologie permet d'avancer étape par étape (red, green, refactoring) via les tests afin de ne faire que le strict nécessaire dans le cadre de l'implémentation d'une spécification. Elle a l'avantage de mettre en place une discipline et un cadre afin d'avoir un code propre, simple (pas de superflu), mais aussi d'avoir une documentation dynamique des spécifications (étant donné que les tests permettent de tester des spécifications, on peut dans ce cas considérer que les tests sont une documentation dynamique). Le **TDD** permet également d'avoir un meilleur taux de couverture mais ce n'est que la conséquence de son application.

- Le clean code :

  Le **clean code** désigne un code clair, simple et facile à comprendre pour n'importe quel développeur. Il évite les mauvaises pratiques comme les noms ambigus, les méthodes trop longues, complexes, les duplications inutiles qui alourdissent la compréhension. Un code propre reste minimaliste : moins il y a d'éléments superflus, moins il y a de risques de bugs et plus la maintenance est simple. Enfin, un code propre est entièrement testé et fiable, ce qui garantit sa stabilité dans le temps. En somme, un code propre est un code lisible, simple, non dupliqué et bien testé, conçu pour durer. Pour une amélioration constante de la qualité de code nous pouvons

passer par ce qu'on appelle le refactoring qui est une méthode de développement qui consiste à modifier le code sans changer le comportement afin d'éliminer ce qui pourrait poser problèmes plus tard.

- Veille technologique :

  La veille technologique fait partie intégrante du rôle d'un développeur artisan. Se tenir informé des nouveaux concepts, des design patterns, des architectures ou des bonnes pratiques permet d'améliorer la qualité du code et d'enrichir sa vision technique. Cette démarche d'apprentissage continu renforce la capacité du développeur à proposer des solutions pertinentes et à anticiper les besoins futurs. C'est un investissement personnel qui bénéficie directement au produit et à l'équipe.

J'ai cité quelques exemples de bonnes pratiques respectant la philosophie du software craftsmanship mais il en existe bien d'autres tels que le BDD, les feedbacks, les mutation testing, etc .
Je vous invite à faire vos propres recherches pour approfondir vos connaissances là-dessus.

Rendons au développeur ses titres de noblesse

Le software craftsmanship accorde une importance capitale au rôle du développeur et le met au premier plan dans le processus de la réalisation d'un produit. Il n'est donc plus vu comme un simple exécutant chargé d'écrire du simple code mais comme un professionnel responsable, soucieux de la qualité du produit et de son fonctionnement sur le long terme. Pour cela, le software craftsmanship met l'accent sur plusieurs points mais je n'en citerai que trois qui sont pour moi les plus importantes :

- Responsabilité :

  Dans la philosophie du software craftsmanship, le développeur prend entièrement la responsabilité de ce qu'il produit. Cela signifie que pour lui, implémenter une fonctionnalité n'est pas suffisant. Il s'engage à livrer un code propre, simple, facilement maintenable et qui dure dans le temps. Etre responsable d'un point de vue craft c'est aussi prendre des décisions techniques réfléchies, de refuser les compromis qui peuvent dégrader la qualité du produit et d'alerter quand il le faut. Le développeur devient garant de la qualité du produit, pas seulement de son fonctionnement.

- Expérience :

  L’expérience est la force du développeur. Quand on parle d'expériences on pense aux années travaillées mais c'est aussi le fait d'apprendre de manière continue, d'avoir une curiosité qu'il faut nourrir constamment, de comprendre les enjeux techniques, d’anticiper les problèmes et de proposer des solutions adaptées. Le software craftsmanship met l'accent sur la montée en compétence permanente : pratiquer, expérimenter, se remettre en question, découvrir de nouvelles approches, améliorer ses méthodes. Plus l’expérience grandit, plus le développeur devient capable de concevoir des architectures solides et d'améliorer son code avec un code de meilleure qualité et de partager ses acquis avec les autres membres de l’équipe.

- Partage/Transmission de connaissances :

  Un développeur ne garde pas ses connaissances pour lui. Le partage est une partie de la culture du software craftsmanship. Il existe plusieurs façons de transmettre son savoir. Cela peut passer par le pair programming, le mob programming, les revues de code, les feedbacks, les ateliers etc. Transmettre ses connaissances permet de faire monter en compétence l’équipe, de réduire les dépendances à une seule personne et de créer une dynamique autour du partage. C'est aussi un moyen de renforcer la cohésion et de faire progresser l’ensemble de l'équipe et ne sera que bénéfique pour le projet.

Conclusion

Le software craftsmanship est une philosophie qui met l'accent sur la qualité technique dans l'objectif d'avoir un produit de qualité, fiable et qui dure dans le temps. Il se concentre également sur l'humain en valorisant son expertise, son professionnalisme et le travail collectif. La philosophie craft s'inscrit dans la continuité de l'agilité afin de compléter cette dernière sur les aspects techniques et humains dans le cadre du développement d'un produit.

Sources :

https://artisandeveloppeur.fr/software-craftsmanship/
https://refactoring.guru/
https://manifesto.softwarecraftsmanship.org/
livre Software craft : TDD, Clean Code et autres pratiques essentielles