



## Plateforme d'intégration continue

---

- Il paraît que ça compile automatiquement ?
- Oui, mais si tu casses, tu amènes les croissants le lendemain.

## Tests unitaires automatisés

---

- Tu as testé cette fonction ?
- Oui, c'est bon elle marche.
- Même lorsque  $x = 0$  ?
- Ah non, sauf lorsque  $x = 0$ .
- Et quand  $x = -1$  ?
- Bon ok, j'ai compris, j'y retourne.

## Tests fonctionnels automatisés

---

- Tu as vraiment un cas de test où l'utilisateur se connecte et se déconnecte 153 fois sans rien faire ?
- C'est pas moi qui fais, c'est Selenium.

## Pair programming

---

- Pourquoi vous êtes deux sur le même écran ?
- En n'utilisant qu'un seul ordinateur, on réduit notre empreinte carbone.

## Design patterns

---

- Là j'ai une variable statique qui compte le nombre d'instances et quand ça dépasse 1, j'ai une exception levée qui détruit toutes les instances sauf une. Tu en penses quoi ?
- rtfm singleton ?

## Product owner dédié

---

- Il faut vraiment que je sois là tous les jours, avec vous ? Sur le même plateau que des... développeurs ? Je suis du métier, moi, ça n'est pas dangereux ?
- Oui, il le faut. Prends sur toi.

## Développeur en renfort

---

- On a un nouveau développeur en renfort pour le projet.
- Super, il va nous aider à avancer plus vite !
- Oui, mais il faut d'abord lui expliquer tout le code.
- Ah, euh, désolé, j'ai une réunion qui commence, je suis déjà en retard !

## Sprint additionnel

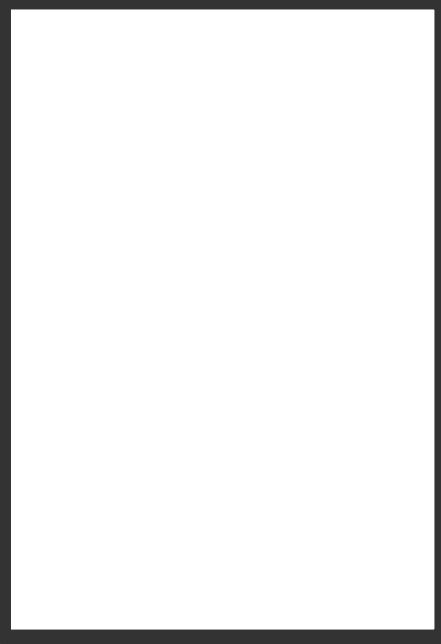
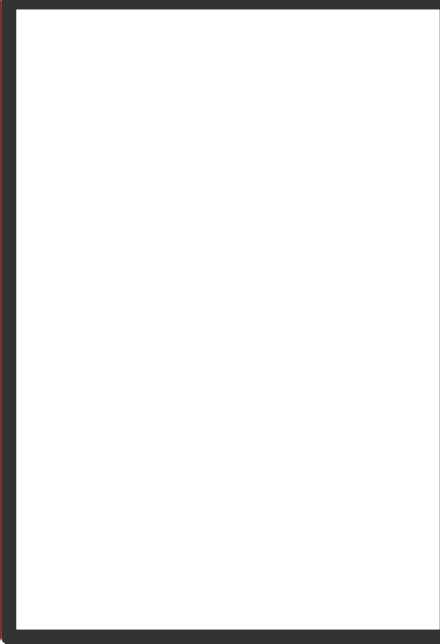
---

- Un sprint de plus, tu crois que ça va passer auprès du client ?
- Mais oui, il suffit d'ajouter un emoji de petit chat penaud à la fin du mail 🐱.

## Déploiement continu

---

- Et là je clique, et hop, le code part en production, automatiquement.
- Et si ça casse ?
- On a aussi automatisé les lettres de licenciement... Je file, j'ai mon profil LinkedIn à mettre à jour !



## Évènement

*Un scandale financier touche nos partenaires bancaires. La Direction de l'Éthique demande de changer urgemment de prestataires de paiement*

### Effet

*Si elle est développée, la fonction « API Prestataires de paiement » retourne dans le backlog et crée une dette technique.*

## Évènement

*La Direction Juridique alerte sur la nécessité absolue d'afficher des informations légales sur le site et vous a envoyé 12 pages à mettre en forme et à ajouter au site.*

### Effet

*On ajoute la carte « Informations légales » au backlog.*

## Évènement

*La Direction de la Sécurité a mené un audit sur le site et a remis un rapport détaillant 38 actions urgentes de mise en conformité.*

### Effet

*On ajoute la carte « Sécurisation du site » au backlog.*

## Évènement

*« On a complètement oublié de mettre en place un système d'évaluations et de commentaires ! Les acheteurs feraient des commentaires positifs et ça convaincrat les indécis d'acheter ! »*

### Effet

*On ajoute l'Epic « Évaluations » au backlog*

## Évènement

*Le Directeur Financier a entendu parler du bitcoin à sa réunion mensuelle du réseau « FinTech et steaks fins » à l'Hôtel du Crillon. Il exige qu'on puisse payer en bitcoins sur le site.*

### Effet

*On ajoute la carte « Paiement en bitcoins » au backlog.*

## Évènement

*La Direction du Développement Durable a fait faire un audit d'accessibilité du site. Il n'est accessible qu'à 45 % ! Vous me corrigez ça, tout de suite !*

### Effet

*On ajoute la carte « Accessibilité » au backlog.*

## Évènement

*Le Directeur de l'Innovation vous demande d'ajouter un chatbot basé sur l'IA générative pour guider les clients potentiels et maximiser vos ventes. Et qu'il puisse s'en vanter dans le Journal du Net et sur son profil LinkedIn.*

### Effet

*On ajoute la carte « Chatbot » au backlog.*

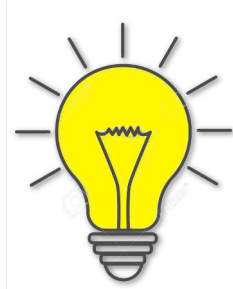
**Plateforme  
d'intégration  
continue**



**Tests  
unitaires  
automatisés**



**Tests  
fonctionnels  
automatisés**



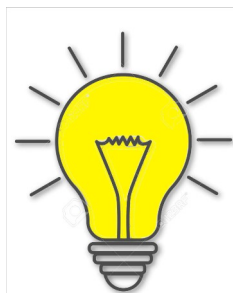
**Pair  
programming**



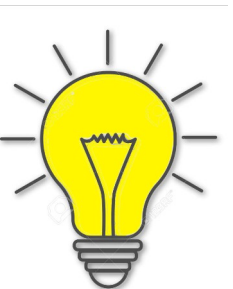
**Design  
patterns**



**Product  
owner dédié**



**Développeur  
en renfort**



**Sprint  
additionnel**



**Déploiement  
continu**



## Tests fonctionnels automatisés

L'**automatisation des tests fonctionnels** permet de vérifier le bon comportement de la fonctionnalité, ou la correction d'une anomalie, dès que le code est écrit.

Ces tests rejoignent au fur et à mesure une base grandissante de tests de non-régression, qu'il est difficile de jouer systématiquement lorsqu'ils sont manuels.

### Effet

Les développeurs ne peuvent plus créer de dette technique lorsqu'ils corrigent une anomalie (ils ne lancent plus le dé).

## Tests unitaires automatisés

Les **tests unitaires automatisés** permettent de vérifier, indépendamment et automatiquement, chaque fonction, micro-service, ou librairie.

Non seulement ils évitent les régressions, mais, pour être réalisables, ils exigent du développeur une meilleure conception du code.

En TDD (**Test Driven Development**), les tests unitaires sont même écrits avant le code.

### Effet

Donner un cache blanc à chaque joueur pour qu'il cache une valeur du dé de son choix. Cette valeur du dé ne créera plus de dette technique.

## Plateforme d'intégration continue

Une **plateforme d'intégration continue** compile le code en temps réel, relève les avertissements, vérifie les règles de codage, fournit des métriques et peut permettre de réaliser des tests automatisés, voire de déployer l'application.

### Effet

Donner un cache blanc à chaque joueur pour qu'il cache une valeur du dé de son choix. Cette valeur du dé ne créera plus de dette technique.

## Product owner dédié

Un **Product Owner dédié** peut s'impliquer bien plus dans le projet, être présent sur le plateau projet, et guider les développeurs au jour le jour sur les besoins métier.

Lorsque ce n'est pas possible, on crée parfois un rôle de **Proxy Product Owner**.

### Effet

Les développeurs créent moins d'anomalies. Avec trois caches blancs, diminuez le nombre d'anomalies créées :





- → 0 anomalie créée
- → 1 anomalie
- → 2 anomalies

## Design patterns

Les **design patterns** sont des solutions de codage, de modélisation ou d'architecture éprouvées face à des problématiques classiques. Leur utilisation rend le code plus maintenable et plus évolutif.

Des design patterns connus : singleton, observateur, factory...

### Effet

La refactorisation du code coûte  au lieu de  . Donnez un cache blanc à chaque développeur pour masquer un .

## Pair programming

En **pair programming**, deux développeurs sont ensemble derrière le même écran. L'un a le clavier, et les deux discutent de la meilleure façon d'implémenter la fonctionnalité.

Le **pair programming** permet de réaliser du code de grande qualité et garantit l'appropriation collective du code.

### Effet

Désormais, deux développeurs peuvent développer ensemble une **User Story**.

Ils payent tous les deux le coût de la **User Story**, mais aucune dette technique n'est créée (pas de jet de dés), et l'on retire une dette technique existante.





## Déploiement continu

Le **déploiement continu** permet d'envoyer automatiquement le nouveau code sur un environnement de recette, voire en production sur une population pilote.

Emblématique de la démarche **DevOps**, le déploiement continu améliore le Time To Market, et permet d'avoir du feedback métier et utilisateurs très rapidement.

### Effet

Au moment de la démonstration, toutes les anomalies apparaissent désormais face visible (elles peuvent donc être corrigées dès le sprint suivant). Sur la carte « Démonstration », remplacez les  par des .

## Sprint additionnel

En agilité, l'arbitrage se fait sur le périmètre, et non sur le délai (ou le coût). On dit parfois que le projet est **time-boxé**.

Négocier un délai face à un client qui a déjà rendue publique une date de livraison n'est pas acceptable.

### Effet

Le client refuse le sprint supplémentaire. Les réunions vous ont fait perdre du temps : un des développeurs confirmés ne produira pas au sprint suivant.

## Développeur en renfort

En agilité, l'arbitrage se fait sur le périmètre, et non sur le coût (ou le délai). On dit parfois que le projet est **cost-boxé**.

De plus, l'arrivée mal préparée d'un renfort en cours de projet désorganise le projet, fait perdre du temps pour la montée en compétences, et ce d'autant plus que le projet est avancé.

### Effet

Le client refuse le surcout induit par le développeur en renfort. Les réunions vous ont fait perdre du temps : un des développeurs confirmés ne produira pas au sprint suivant.