



DIGITAL  
SUMM'R

Jeudi 29 août

Tech / Produit

# La qualité, sans déshabiller le produit pour habiller la tech'

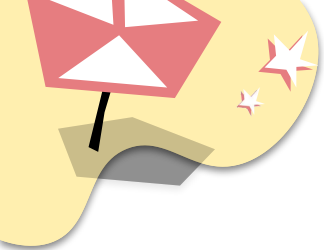
Regards croisés de CTO expérimentés sur le triptyque **Coût-Qualité-Délais** et les pratiques modernes de développement

avec

**Xavier Barry**  
CTPO Freelance



**Kévin Delfour**  
ex CTO, Directeur



# Un peu de théorie pour commencer

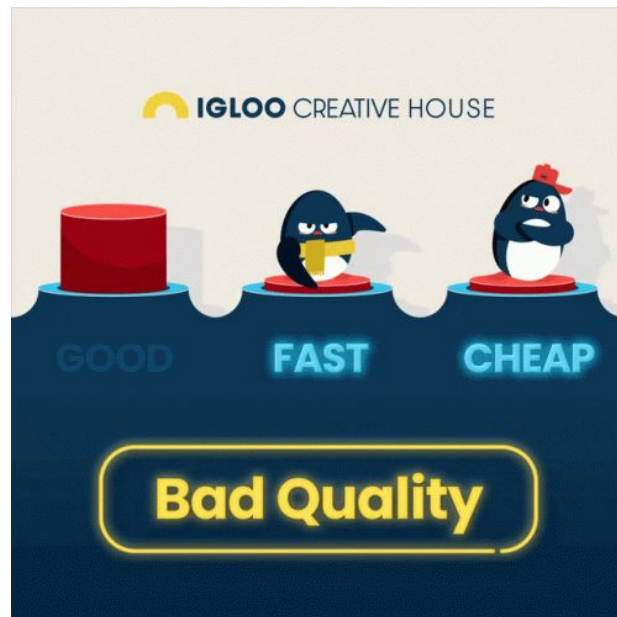
Car ça nous rassure et ça nous met en confiance de commencer par là 😊

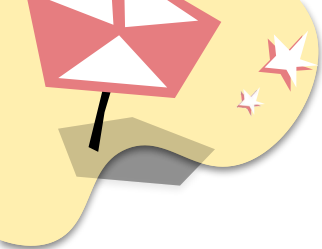
**Et puis on tient à briser vos croyances ! 🌟**

🕒 10 min

# Introduction

- **Contexte** : Le développement logiciel a longtemps été gouverné par le triptyque Coût-Qualité-Délais.
- **Objectif de la session** : Analyser pourquoi ce modèle est dépassé et présenter des pratiques modernes qui équilibrent ces trois aspects sans compromis.





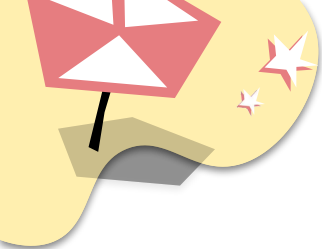
# Présentation du Triptyque Coût-Qualité-Délais

- **Définition :**

-  **Coût** : Ressources financières et humaines nécessaires.
-  **Qualité** : Fonctionnalités, performance, et absence de bugs.
-  **Délais** : Temps nécessaire pour livrer le produit.

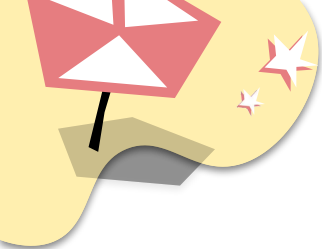
- **Interdépendance :**

Traditionnellement, on considère qu'il faut sacrifier un élément pour en optimiser un autre (par exemple, augmenter les coûts pour améliorer la qualité ou réduire les délais).



## Illustration du Triptyque Coût-Qualité-Délais ▲

- **Exemple 1 :**
  - Un projet avec un budget limité peut devoir sacrifier la qualité ou prolonger les délais.
- **Exemple 2 :**
  - Un projet avec une échéance stricte peut nécessiter des ressources supplémentaires ou accepter une baisse de qualité.
- **Exemple 3 :**
  - Un produit de haute qualité peut nécessiter plus de temps et de budget.



# Pourquoi le Triptyque n'est plus pertinent 💀

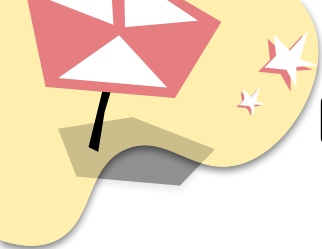


→ Évolution des Méthodes de Développement.

→ Nouvelles Pratiques :

- 1 **Méthodes agiles** : Adaptabilité et itération rapide.
- 2 **DevOps et CI/CD** : Automatisation des tests et des déploiements.
- 3 **Software craftsmanship** : Code propre et maintenable.

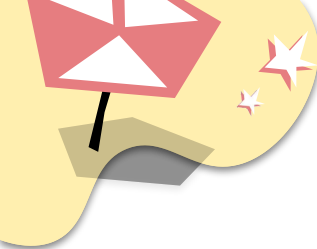




# 1 Impact des Méthodes Agiles

Exemple : Méthodologie Scrum

- **Coût** : Réduction des coûts à long terme grâce à l'amélioration continue.
- **Qualité** : Maintien de la qualité via des revues de sprint et des tests réguliers.
- **Délais** : Respect des délais grâce à des sprints courts et planifiés.

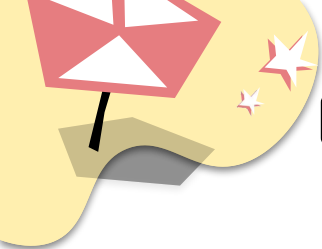


## 2 DevOps et CI/CD

Exemple : Pipeline CI/CD

- **Coût** : Investissement initial dans l'automatisation, économies à long terme.
- **Qualité** : Amélioration de la qualité avec des tests automatisés.
- **Délais** : Réduction des délais de livraison avec des déploiements continus.

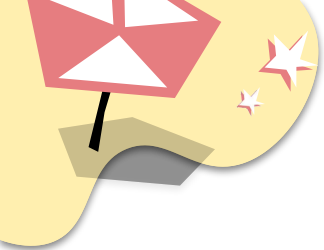




## ③ Software Craftsmanship

Exemple : Principes du Clean Code

- **Coût** : Effort initial pour écrire du code propre, réduction des coûts de maintenance.
- **Qualité** : Haute qualité grâce à des pratiques de codage rigoureuses.
- **Délais** : Facilitation des modifications rapides et efficaces.



# Mise en situation

Car la théorie c'est bien mais les exemples réels c'est mieux 😊

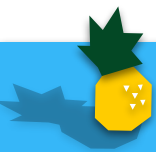
🕒 15 min





## Sacrifier la qualité pour respecter les délais

- **Situation** : Sous la pression de respecter une date de lancement imminente, l'équipe technique réduit les cycles de test pour accélérer le développement.
- **Impact** : Le produit est lancé avec des défauts majeurs, affectant la réputation de l'entreprise et entraînant une perte de confiance chez les clients.
- **Solutions** :
  - **Planification réaliste** : Établir des délais réalistes dès le début pour éviter la pression de couper les coins ronds en matière de qualité.
  - **Priorisation agile** : Adopter une méthodologie agile pour ajuster les priorités en fonction des contraintes de temps sans compromettre la qualité.
  - **Réserves de temps pour la qualité** : Allouer du temps spécifique pour le peaufinage et les tests de qualité dans le calendrier du projet.





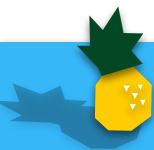
# Ignorer les retours des utilisateurs

- **Situation** : L'équipe produit décide de lancer une nouvelle fonctionnalité sans tenir compte des retours des utilisateurs obtenus lors des phases de test, pensant que les préoccupations soulevées ne sont pas significatives.
- **Impact** : Après le lancement, il s'avère que la fonctionnalité est peu utilisée ou génère de la frustration, nécessitant des corrections coûteuses et des excuses publiques.
- **Solutions** :
  - **Processus de feedback intégré** : Mettre en place un système structuré pour collecter et analyser les feedbacks des utilisateurs à toutes les étapes du développement.
  - **Décisions basées sur les données** : Utiliser des données quantitatives et qualitatives pour prendre des décisions éclairées concernant les modifications du produit.
  - **Révisions itératives** : Planifier des itérations de développement pour intégrer les feedbacks et améliorer continuellement le produit.



## Tests insuffisants en environnement réel

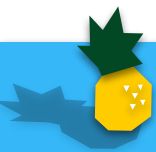
- **Situation** : Les tests sont réalisés uniquement dans des environnements contrôlés qui ne reflètent pas les conditions réelles d'utilisation.
- **Impact** : Une fois en production, le produit ne fonctionne pas correctement sous différentes configurations ou charges, menant à des pannes et à la frustration des utilisateurs.
- **Solutions** :
  - **Tests en environnement de production simulé** : Créer des environnements de test qui imitent le plus fidèlement possible les conditions réelles d'utilisation.
  - **Monitoring continu** : Mettre en place un monitoring continu pour détecter et résoudre rapidement les problèmes qui ne se manifestent qu'en production.
  - **Tests de charge et de stress** : Effectuer régulièrement des tests de charge et de stress pour évaluer la performance sous différentes charges d'utilisation.

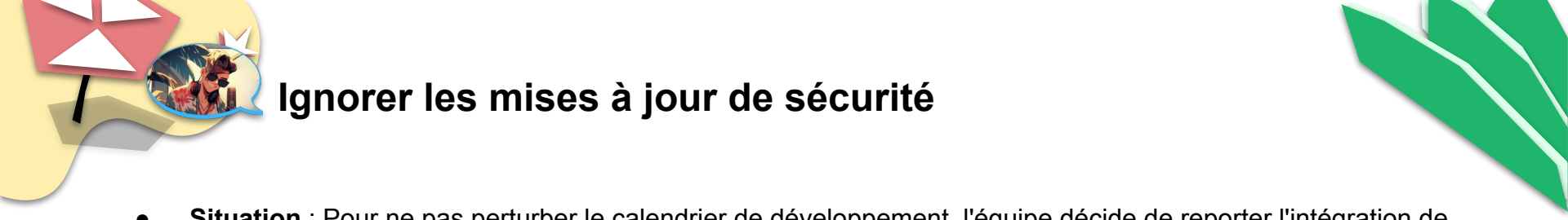




# Manque de communication entre les équipes

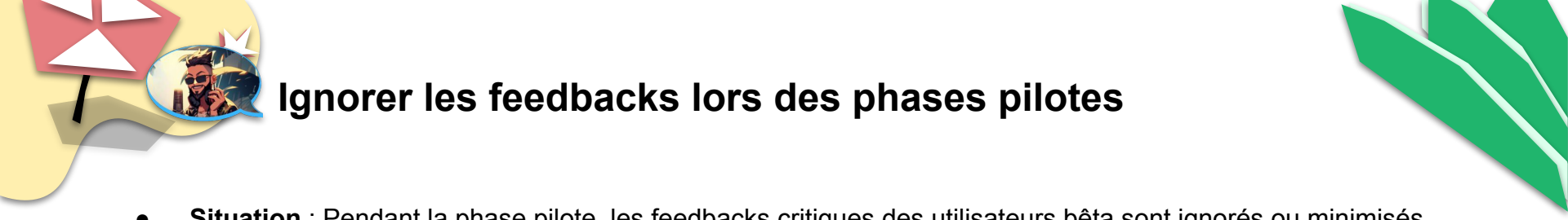
- **Situation** : Les équipes de développement et de produit travaillent en silos, sans communication régulière ou partage d'objectifs communs.
- **Impact** : Des malentendus sur les spécifications du produit conduisent à un développement qui ne répond pas aux attentes du marché, nécessitant des révisions majeures après le lancement.
- **Solutions** :
  - **Réunions de synchronisation régulières** : Organiser des réunions régulières entre les équipes pour garantir que tout le monde est aligné sur les objectifs et les attentes.
  - **Outils de collaboration** : Utiliser des outils de collaboration et de gestion de projet pour maintenir une communication fluide et documentée.
  - **Formation interfonctionnelle** : Encourager les membres des différentes équipes à comprendre les rôles des autres pour renforcer l'unité et la collaboration.





# Ignorer les mises à jour de sécurité

- **Situation** : Pour ne pas perturber le calendrier de développement, l'équipe décide de reporter l'intégration de mises à jour de sécurité critiques.
- **Impact** : Le produit subit une violation de données, exposant les informations sensibles des clients et entraînant des conséquences juridiques et financières graves.
- **Solutions** :
  - **Politique de sécurité stricte** : Établir une politique stricte qui exige la mise à jour régulière des logiciels et des bibliothèques pour la sécurité.
  - **Automatisation des mises à jour** : Mettre en place des outils automatiques pour appliquer les mises à jour de sécurité dès qu'elles sont disponibles.
  - **Audits de sécurité réguliers** : Conduire des audits de sécurité périodiques pour identifier et rectifier les vulnérabilités avant qu'elles ne soient exploitées.



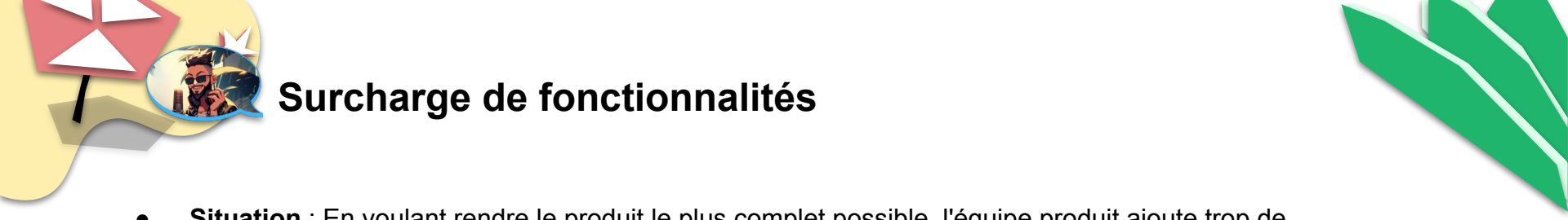
# Ignorer les feedbacks lors des phases pilotes

- **Situation** : Pendant la phase pilote, les feedbacks critiques des utilisateurs bêta sont ignorés ou minimisés car l'équipe est trop attachée à leur vision initiale du produit.
- **Impact** : Le produit final ne répond pas aux besoins des utilisateurs, entraînant des révisions coûteuses après le lancement et des opportunités de marché manquées.
- **Solutions** :
  - **Processus de feedback structuré** : Établir un processus clair pour collecter, analyser et intégrer les feedbacks pendant les phases pilotes.
  - **Réunions régulières d'évaluation** : Organiser des réunions régulières avec l'équipe projet pour discuter des feedbacks et ajuster le développement en conséquence.
  - **Flexibilité dans la planification** : Planifier le développement de manière à permettre des ajustements basés sur les feedbacks des utilisateurs sans perturber le calendrier global.



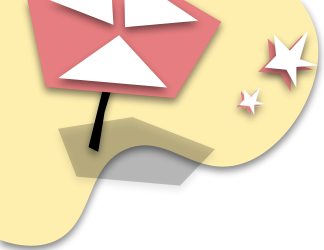
## Déploiement sans plan de repli

- **Situation** : Une mise à jour majeure est déployée rapidement pour avoir une nouvelle fonctionnalité en production sans avoir testé un plan de repli en cas d'échec du déploiement.
- **Impact** : Lorsque des problèmes critiques surviennent après le déploiement, l'équipe n'a pas de solution rapide pour revenir à la version antérieure, causant des interruptions prolongées pour les utilisateurs.
- **Solutions** :
  - **Plans de repli et de rollback** : Développer et tester des plans de repli pour chaque mise à jour majeure afin de pouvoir revenir en arrière rapidement en cas de problème.
  - **Tests de rollback** : Tester systématiquement les procédures de rollback pour s'assurer qu'elles fonctionnent correctement avant le déploiement.
  - **Monitoring post-déploiement** : Mettre en place un monitoring intensif après les déploiements pour détecter rapidement les problèmes et intervenir si nécessaire.



# Surcharge de fonctionnalités

- **Situation** : En voulant rendre le produit le plus complet possible, l'équipe produit ajoute trop de fonctionnalités sans évaluation critique, rendant l'interface utilisateur complexe et difficile à utiliser.
- **Impact** : Les utilisateurs sont confus par le produit, l'adoption est faible, et des ressources importantes sont gaspillées sur des fonctionnalités peu utilisées.
- **Solutions**:
  - **Priorisation basée sur la valeur utilisateur** : Utiliser des méthodes de développement Agile pour prioriser les fonctionnalités qui apportent le plus de valeur aux utilisateurs.
  - **Tests d'utilisabilité réguliers** : Organiser des sessions de tests avec de vrais utilisateurs pour obtenir des retours sur la complexité et l'utilité des fonctionnalités.
  - **MVP (Produit Minimum Viable)** : Se concentrer sur le lancement d'un MVP avec des fonctionnalités essentielles avant d'ajouter d'autres fonctionnalités basées sur les feedbacks des utilisateurs.



## En résumé

Pour ne rien oublier ...

## votre antisèche !

Tech / Produit

**La qualité, sans déshabiller le  
produit pour habiller la tech'**

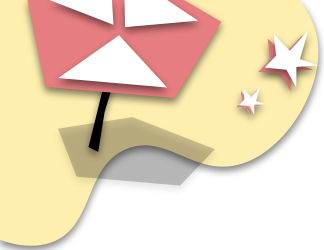


**Xavier Barry**  
CTPO Freelance

**Kévin Delfour**  
ex CTO, Directeur

## Les 5 choses à retenir du talk :

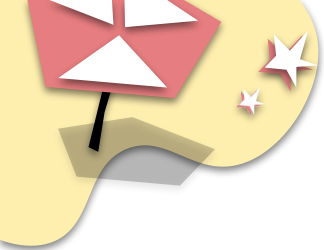
1. **Le triptyque Coût-Qualité-Délais est dépassé** : Les pratiques modernes montrent qu'il est possible d'équilibrer ces trois aspects sans compromis.
2. **L'agilité et l'amélioration continue favorisent l'équilibre** : Les méthodes agiles permettent d'adapter rapidement les priorités tout en maintenant la qualité et en maîtrisant les coûts.
3. **DevOps et automatisation réduisent les erreurs et les coûts** : L'automatisation des tests et des déploiements assure une qualité constante et réduit les coûts à long terme.
4. **Le Software Craftsmanship est essentiel pour la qualité** : Un code propre et maintenable est un investissement qui réduit les coûts de maintenance et facilite les évolutions futures.
5. **La flexibilité est la clé de la réussite** : Adopter des pratiques flexibles et adaptatives permet de répondre aux besoins changeants sans sacrifier la qualité ou les délais.



# Merci pour votre attention

Des questions ?





# Annexes

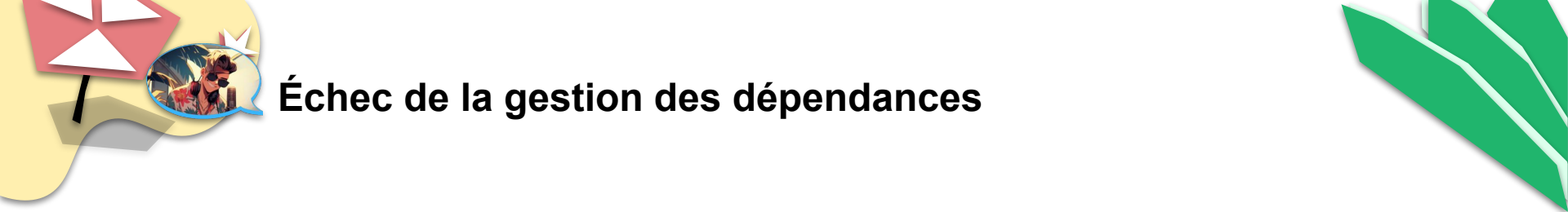
D'autres mises en situations qu'on garde sous le coude 🤖



# Ne pas prioriser l'accessibilité



- **Situation** : L'équipe de produit décide de ne pas rendre la nouvelle application web accessible aux personnes handicapées pour réduire les coûts.
- **Impact** : Non seulement l'entreprise exclut une partie significative de la population, mais elle viole également les réglementations légales sur l'accessibilité, risquant des amendes et une mauvaise publicité.
- **Solutions** :
  - **Normes d'accessibilité dès la conception** : Intégrer les normes d'accessibilité dès le début du processus de conception pour s'assurer que le produit est utilisable par tous.
  - **Tests d'accessibilité réguliers** : Effectuer des tests d'accessibilité régulièrement pour identifier et corriger les problèmes d'accessibilité au fur et à mesure du développement.
  - **Formation et sensibilisation à l'accessibilité** : Former et sensibiliser les équipes de développement et de produit sur l'importance de l'accessibilité et les meilleures pratiques associées.



# Échec de la gestion des dépendances

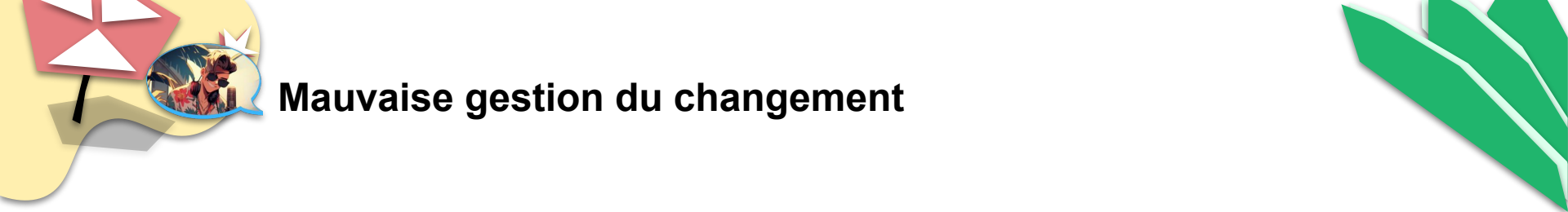
- **Situation** : L'équipe de développement néglige de gérer correctement les dépendances de leur projet, en utilisant des versions obsolètes ou non sécurisées de bibliothèques.
- **Impact** : Des failles de sécurité sont exploitées et le produit souffre de problèmes de performance, entraînant des critiques négatives et des coûts de maintenance élevés.
- **Solutions**:
  - **Audit régulier des dépendances** : Effectuer des audits de sécurité et de compatibilité réguliers sur toutes les dépendances pour s'assurer qu'elles sont à jour et ne présentent pas de risques de sécurité.
  - **Utiliser des outils de gestion des dépendances** : Implémenter des outils automatiques qui peuvent gérer et mettre à jour les dépendances de manière proactive.
  - **Formation continue** : Former les développeurs sur l'importance de la gestion des dépendances et les meilleures pratiques associées



# Délaisser l'optimisation de performance



- **Situation** : En se concentrant trop sur l'ajout de nouvelles fonctionnalités, l'équipe technique néglige l'optimisation des performances existantes.
- **Impact** : Le produit devient lent et inefficace, ce qui nuit à l'expérience utilisateur et augmente le taux de désabonnement ou de mécontentement.
- **Solutions** :
  - **Suivi des performances** : Implémenter des outils de suivi des performances pour détecter et corriger les problèmes dès leur apparition.
  - **Revue de code régulière** : Organiser des revues de code régulières pour identifier et optimiser les parties du code qui peuvent ralentir le système.
  - **Tests de performance réguliers** : Intégrer les tests de performance dans le cycle de vie du développement pour garantir que les nouvelles fonctionnalités n'affectent pas négativement les performances générales du produit.



# Mauvaise gestion du changement

- **Situation** : Les mises à jour du produit sont déployées sans informer adéquatement les utilisateurs ni les former sur les nouvelles fonctionnalités.
- **Impact** : Les utilisateurs sont surpris et déroutés par les changements, ce qui peut entraîner une baisse de l'engagement et de la satisfaction des clients.
- **Solutions** :
  - **Communication proactive** : Informer les utilisateurs bien à l'avance des mises à jour à venir et de leurs impacts potentiels.
  - **Formation des utilisateurs** : Offrir des formations et des ressources d'apprentissage pour aider les utilisateurs à s'adapter aux nouvelles fonctionnalités.
  - **Feedback post-déploiement** : Collecter activement les retours des utilisateurs après chaque mise à jour pour évaluer l'efficacité de la gestion du changement.