Ha Zut !
Le DevOps a mangé ma vélocité

Jean-Marc Lavoie (SingularIT Solutions)
Sylvie Trudel (UQAM)

SingularIT Solutions

UQÀM

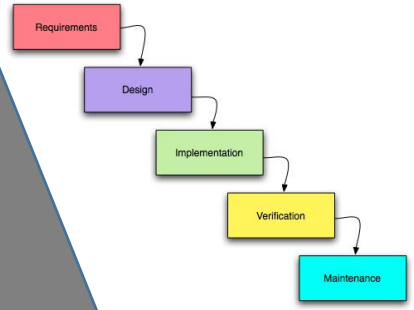
Le vélocité,

c’est quoi?

La vélocité, c'est ...

≈ Productivité = Efficacité + Efficience

En Waterfall,
«Productivité de livraison du logiciel»:
Quantité de logiciel / mois-personne



?

- 4 sem. de 35h = 140h
- 4,3 sem. de 40h = 172h

→ Variable !

En Agile,

Vélocité
=
USP / Sprint

Subjectif!
Non reproductible

2, 3 ou 4 semaines ?
→ Variable !

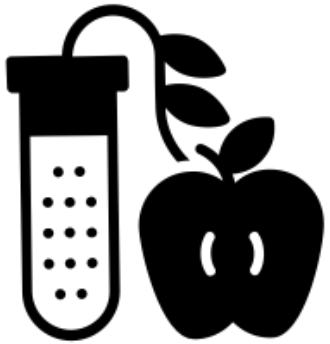
Selon Larousse: « Grande rapidité dans le mouvement »

En DevOps, la vélocité ça pourrait être…

- *Comme en Waterfall: Quantité de logiciel / mois-personne → Variable!*
- *Comme en Agile: # USP / Sprint → Variable!*

- *Quelques équipes utilisent le temps de cycle moyen (de Lean/Kanban)*

- *Différences importantes de l'effort et la durée, entre:*



Retour du terrain sur la vélocité

Résultat attendu Agile et DevOps

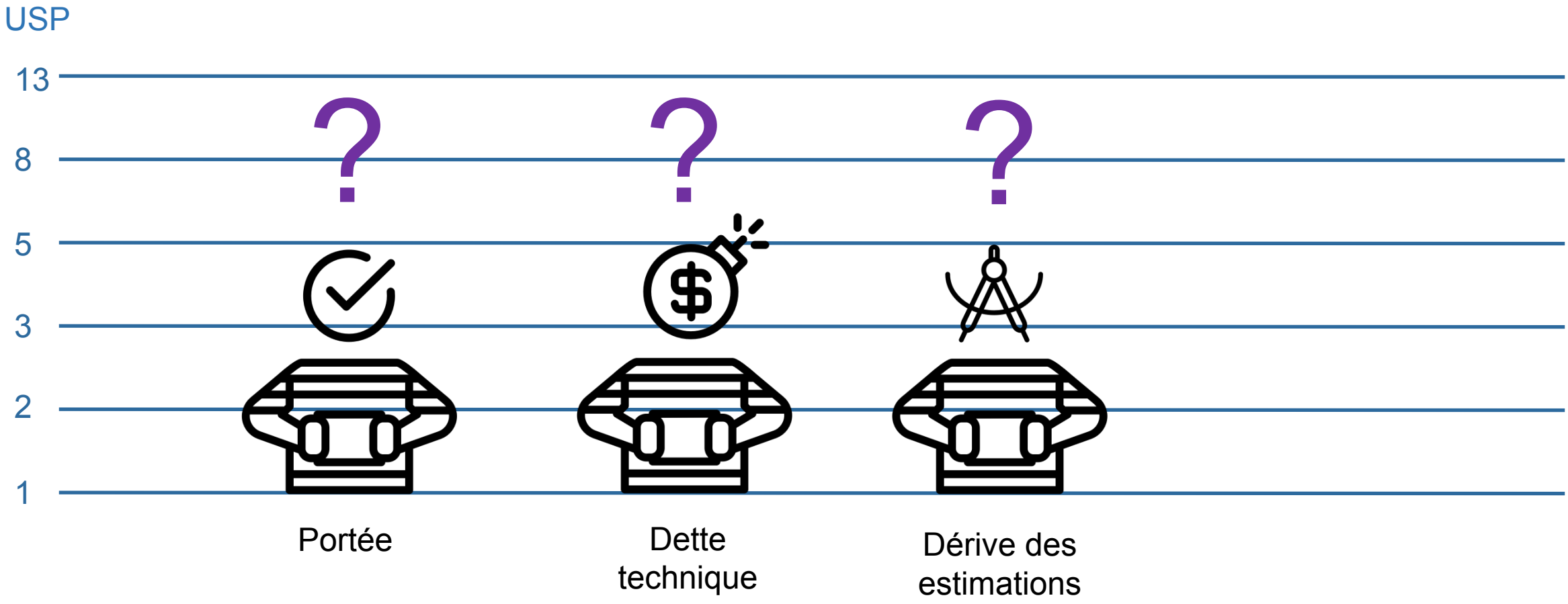
- *Augmentation de notre vélocité avec le temps:*
  - *Bonnes rétrospectives*
  - *+ de réutilisation*
  - *+ de maturité de l'équipe*
  - *+ compréhension du domaine d'affaires et des besoins*

Résultat obtenu

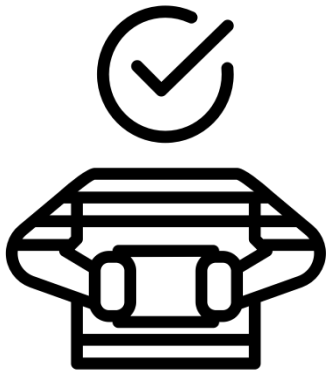
- *En Agile:*
  - *Tendance à stagner*
  - *Parfois ça diminue*
- *En DevOps:*
  - *Tendance générale à diminuer avec le temps!*

Mais qui a volé la vélocité?

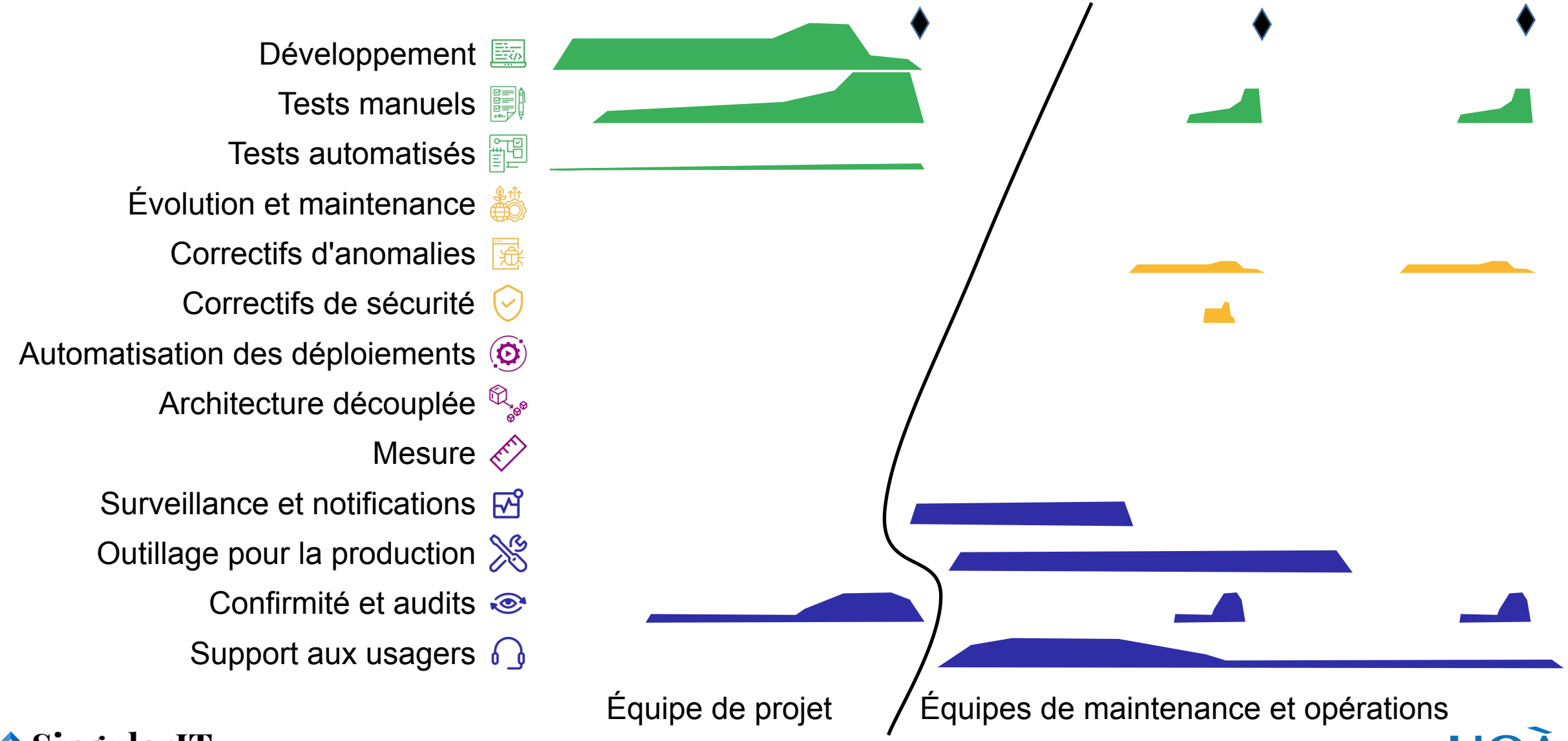
Baisse de vélocité en DevOps? Quel est le coupable?



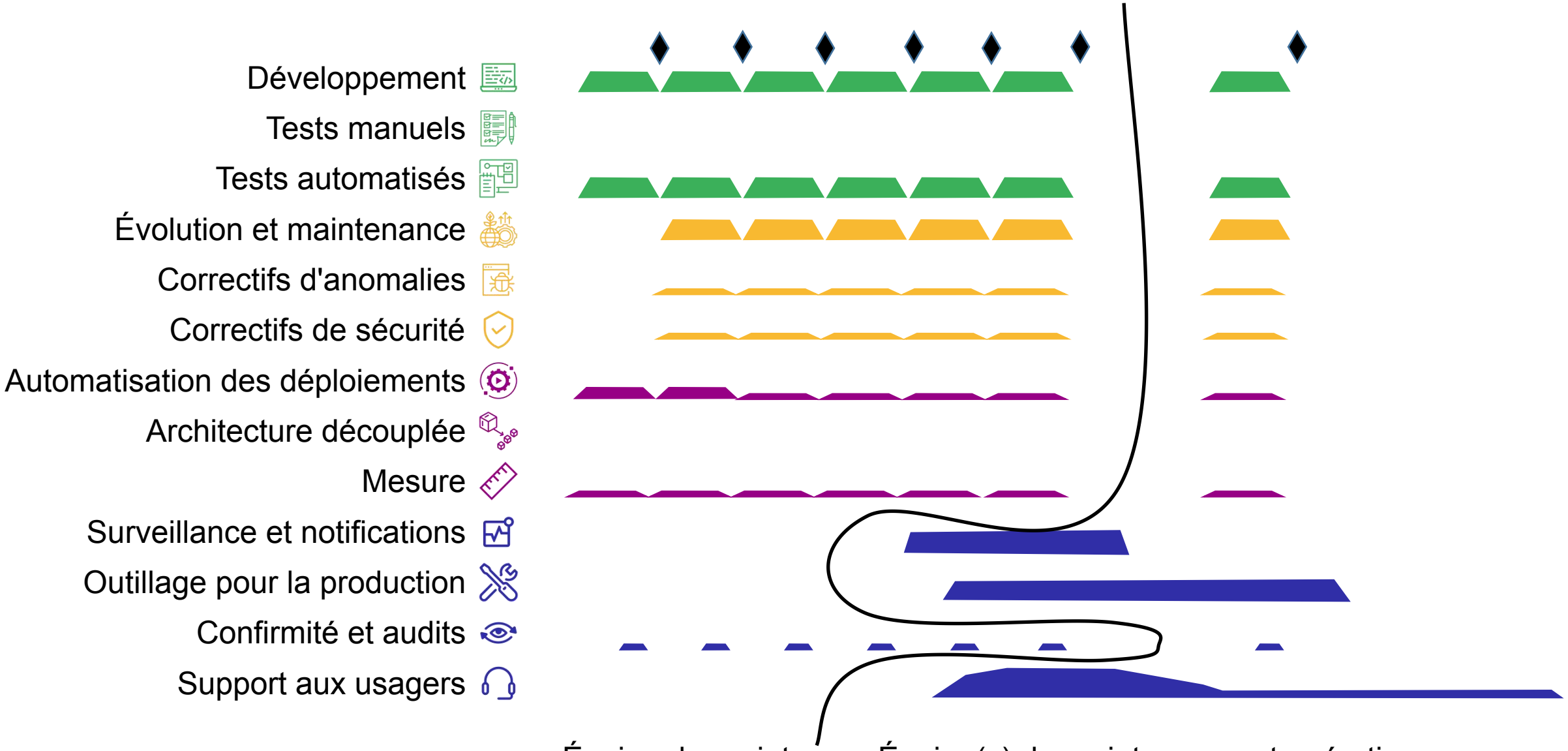
Parlons de la portée!



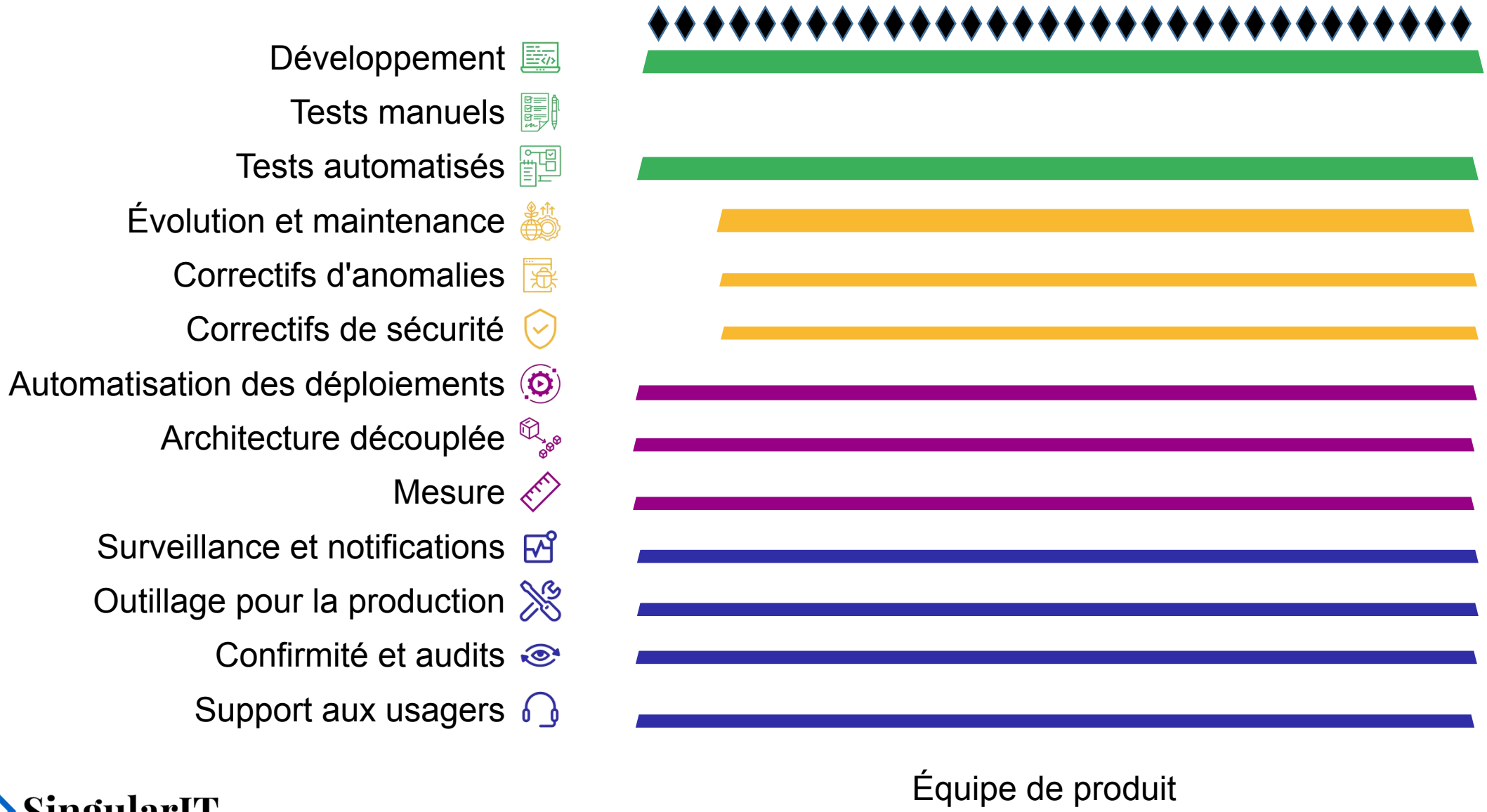
Activités en Waterfall classique



Activités en Agile



Activités en DevOps



Grande portée == petite vélocité ?

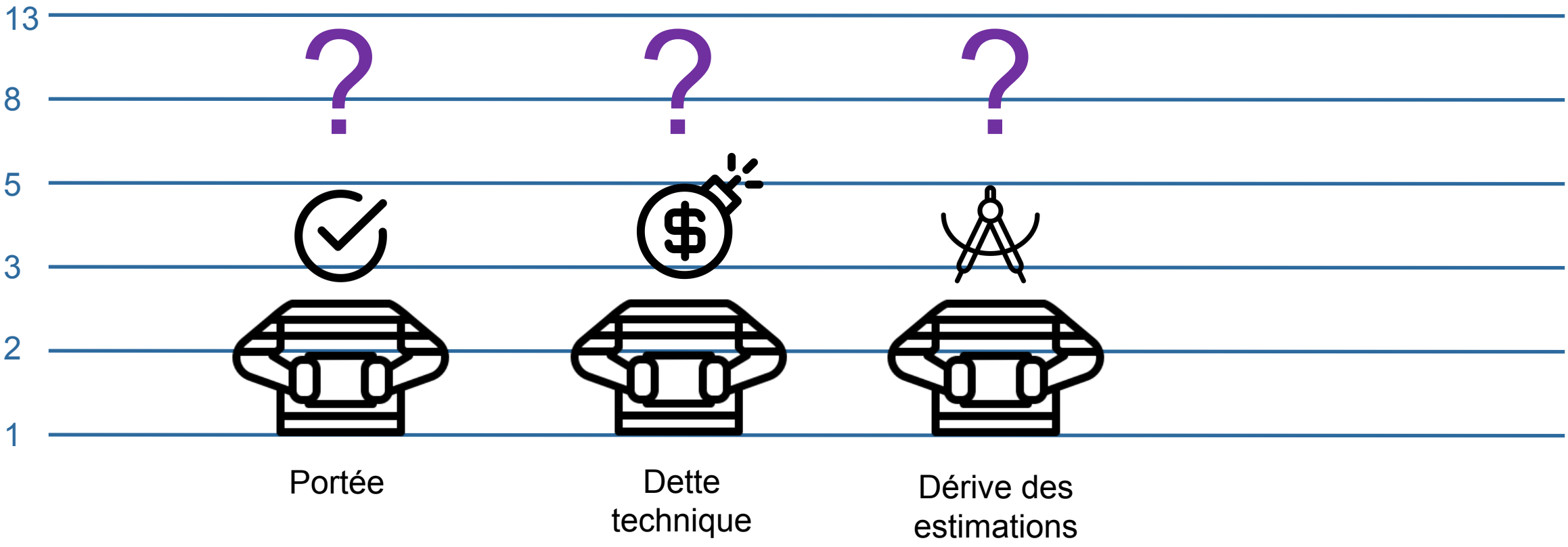
En Agile et en DevOps:

- *Automatisation : accélère la livraison*
- *Tests automatisés : favorise la qualité*
- *Meilleure qualité : favorise la vélocité*

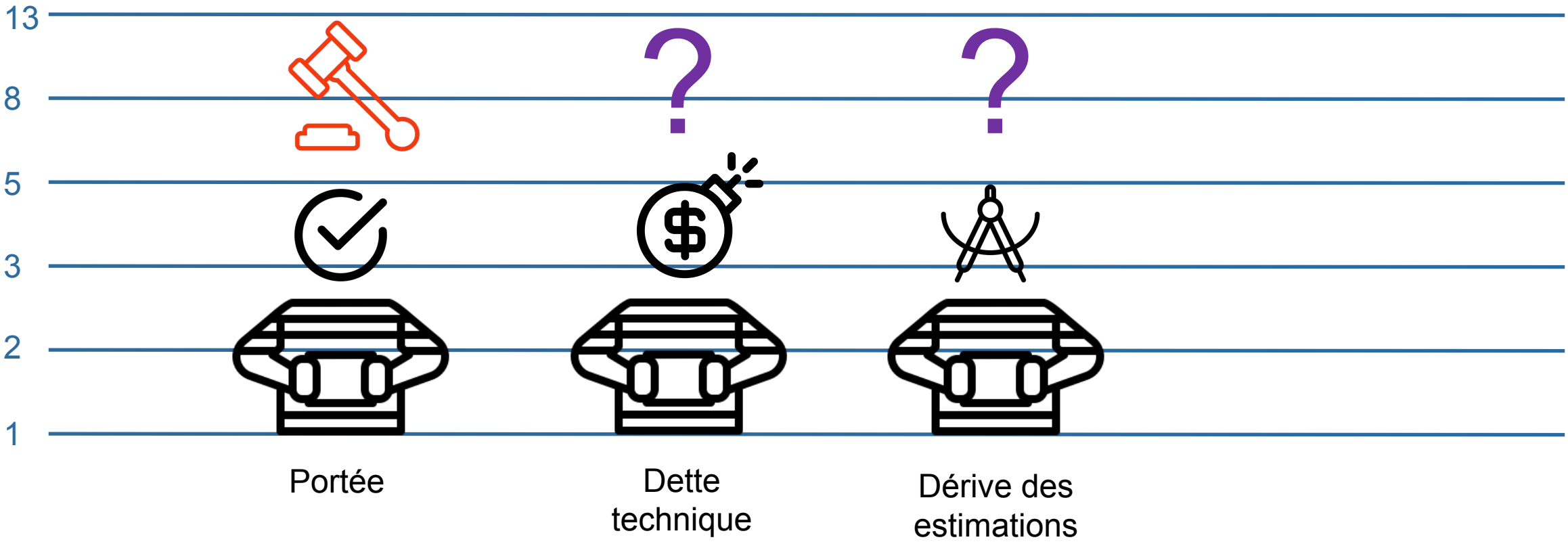
Mais en DevOps:

- *Supporter la production demande de l'effort*

La portée est-elle coupable ?



Coupable !



Les causes de l'augmentation de la portée

- *En DevOps, beaucoup de travaux qui appartenaient aux opérations*

Gestion des données de test

Intégrer la sécurité dès le départ

Surveillance et observabilité

Développement à branche unique

Gestion du changement de base de données

Automatisation des déploiements

Tests continus

Infrastructure cloud (Infonuagique)

Livraison continue

Architecture (découplée)

Gestion du code (qualité, réutilisable, maintenable)

Notification proactive des échecs

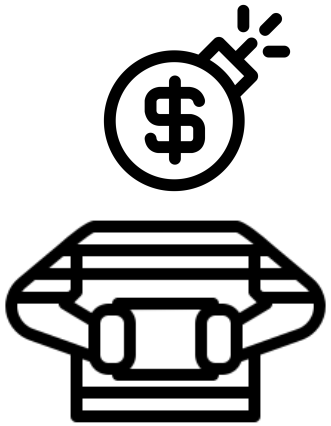
Système de surveillance permettant d'éclairer les décisions métiers

Solution → pour mieux gérer la portée

- *Ajuster la définition de terminé*
  - *Inclure les tâches d'opérations*
    → Qu'elles soient VISIBLES afin d'être estimées!

- *Avoir des membres d'équipe experts en opérations*
  - *Parce qu'ils savent ce qui est nécessaire, alors on évite de les oublier!*
  - *Ça va plus vite aussi!*
  - *Équipe multidisciplinaire: les Ops doivent en faire partie!*
  - *Possibilité: équipe un peu plus grosse…*

Parlons de la dette technique!



Pensez-vous qu'il s'agit d'une dette technique ...

Fonctionnalité manquante ?

Vulnérabilité de sécurité (découverte en 2021) ?

Absence de surveillance et notification ?

Définition

*« Dans les logiciels d'envergure, la dette technique consiste en un ensemble de **conceptions** et **d'implémentations** plus **rapides** originalement, mais qui mettent en place un contexte technique qui peut rendre un changement futur plus **couteux** ou **impossible**.*

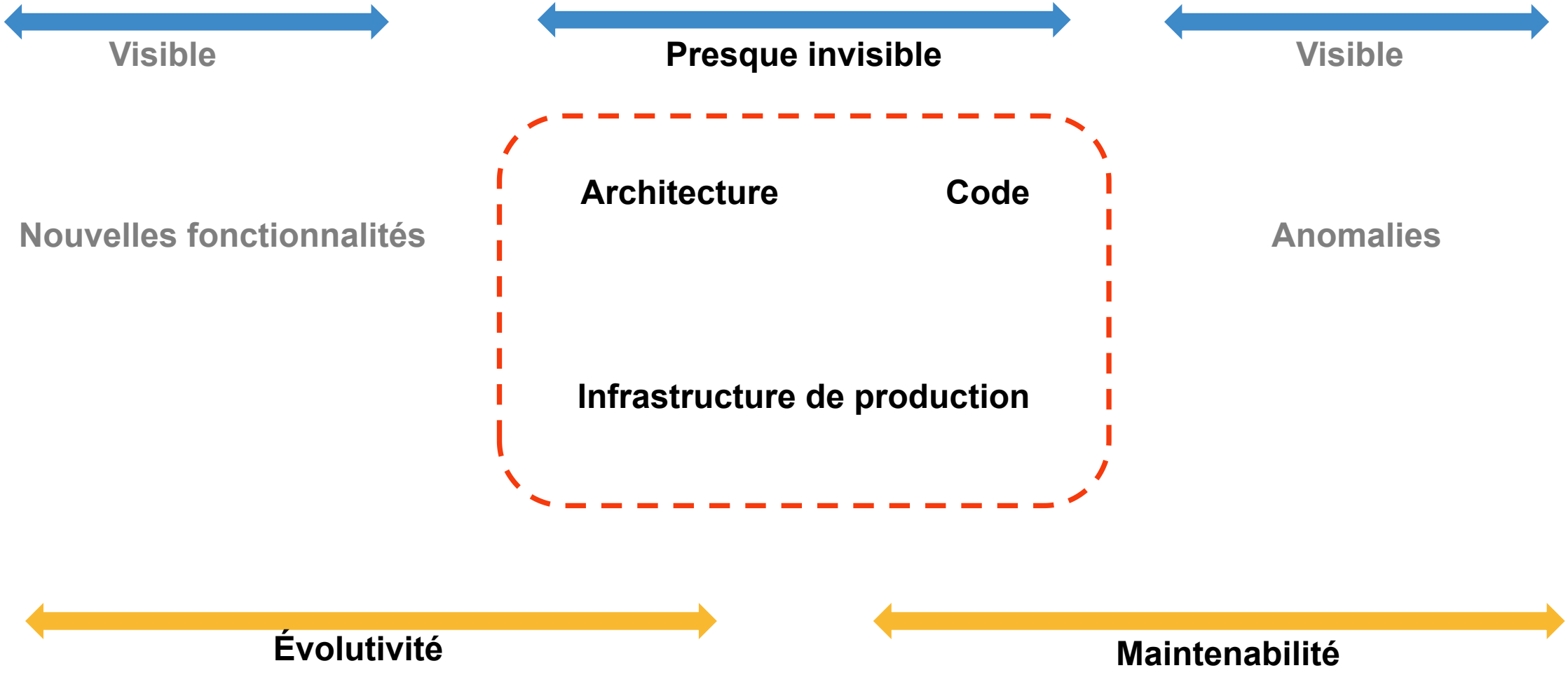
*Il s'agit d'une **obligation** potentielle de réaliser des **travaux** dont les impacts sont limités aux caractéristiques **internes** du système, principalement, mais non uniquement la **maintenabilité** et **l'évolutivité**. »*

- Philippe Kruchten, Ipek Ozkaya

Ou plus simplement...

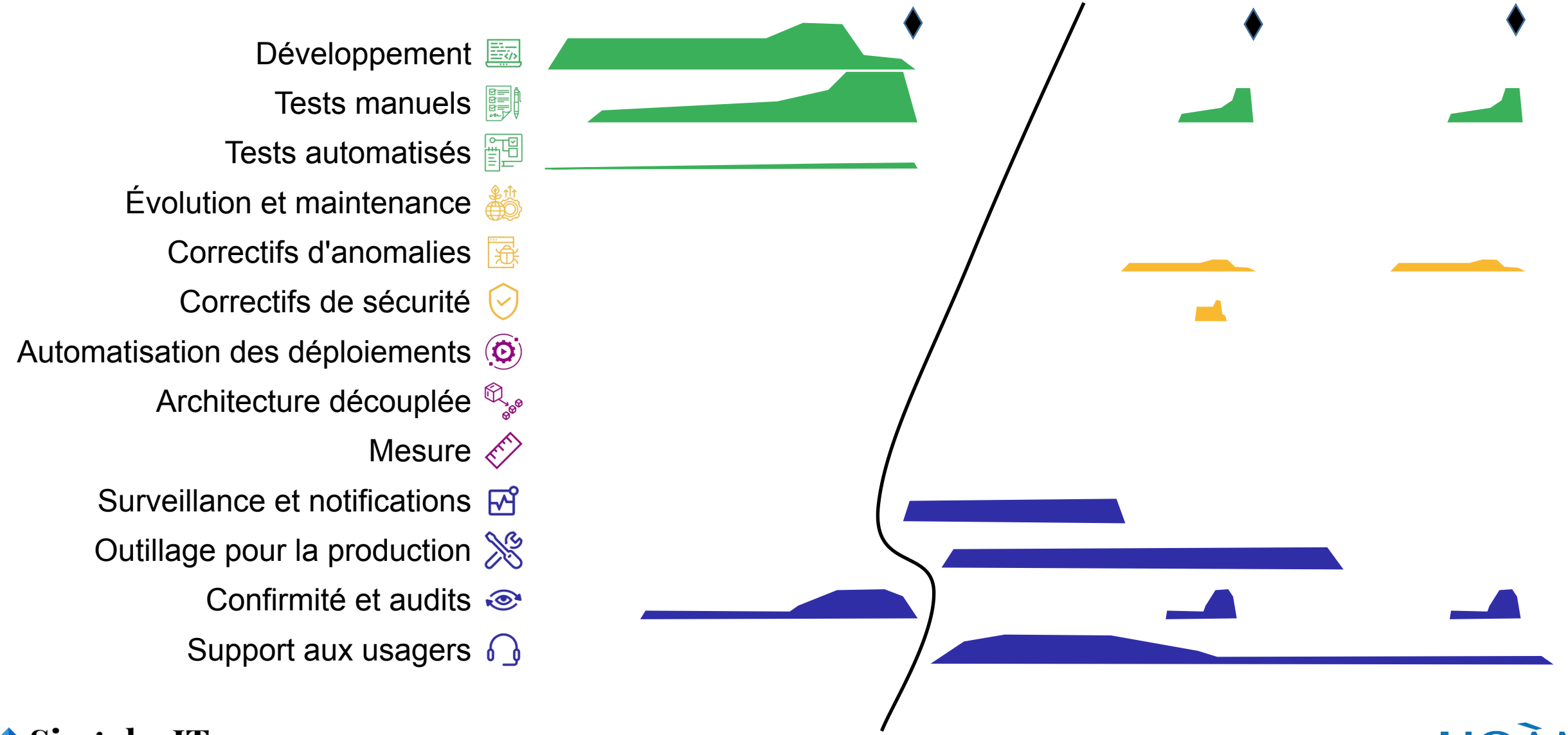
- *Prendre des raccourcis*
- *Pelleter par en avant*
- *Faire une première version techniquement minimaliste*
- *Solution technique qui n'est plus adéquate parce que les besoins ont changés*

Paysage de la Dette Technique

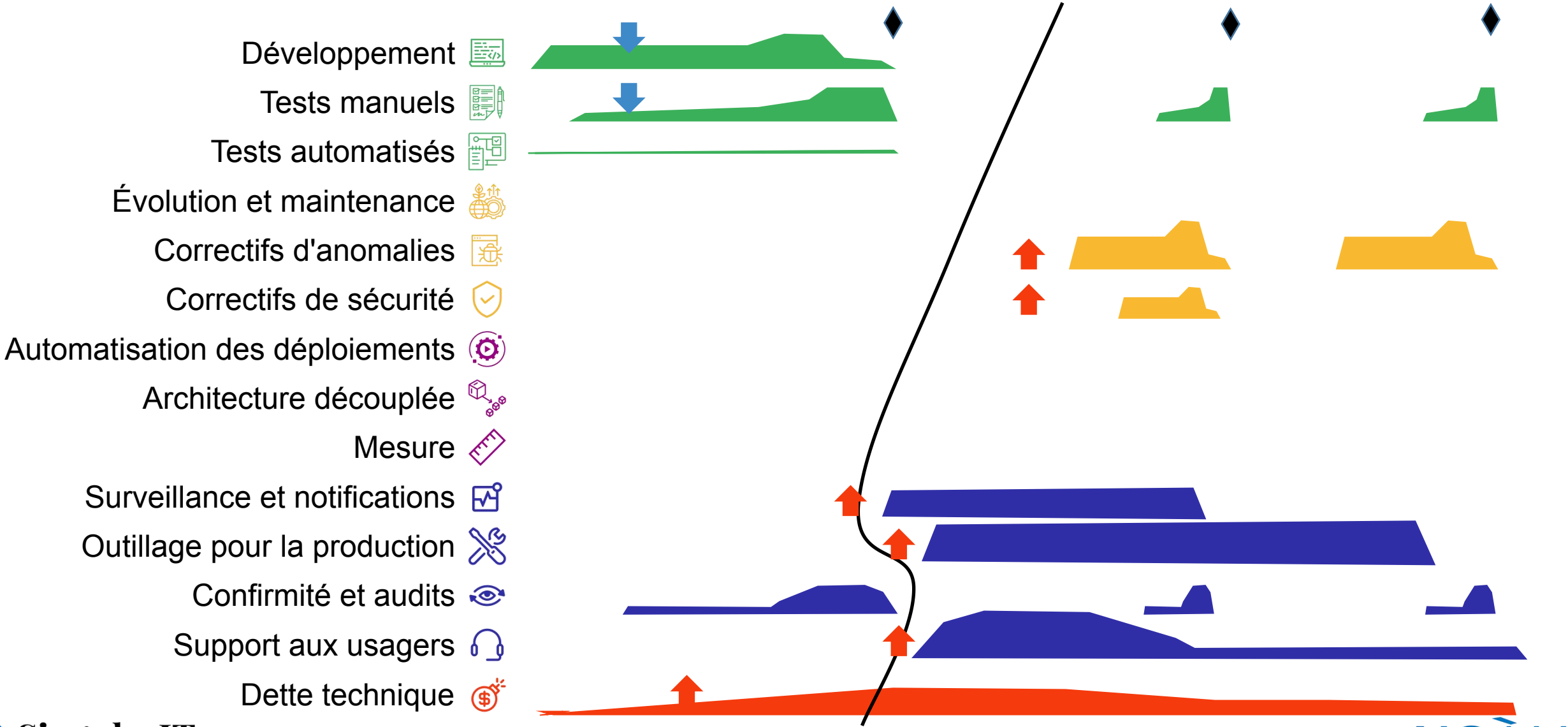


Source: Managing Technical Debt: Reducing Friction in Software Development (SEI Series in Software Engineering) 1st Edition, Philippe Kruchten, Ipek Ozkaya

Activités en Waterfall classique

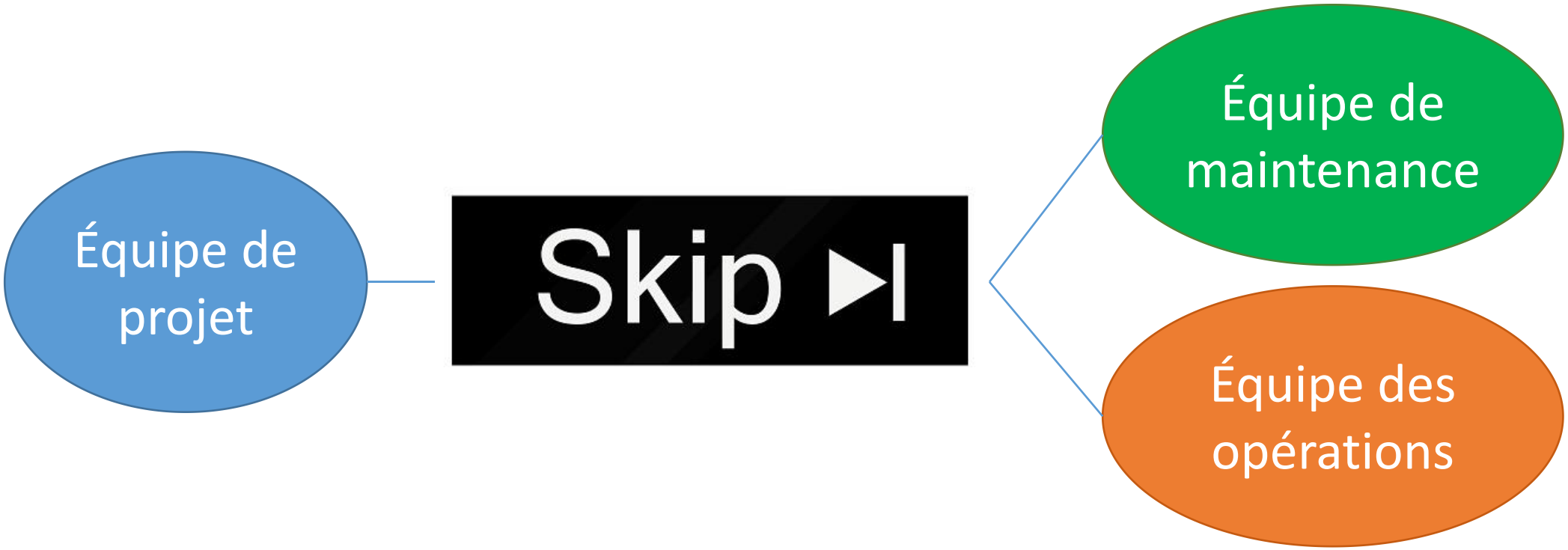


Activités en Waterfall classique

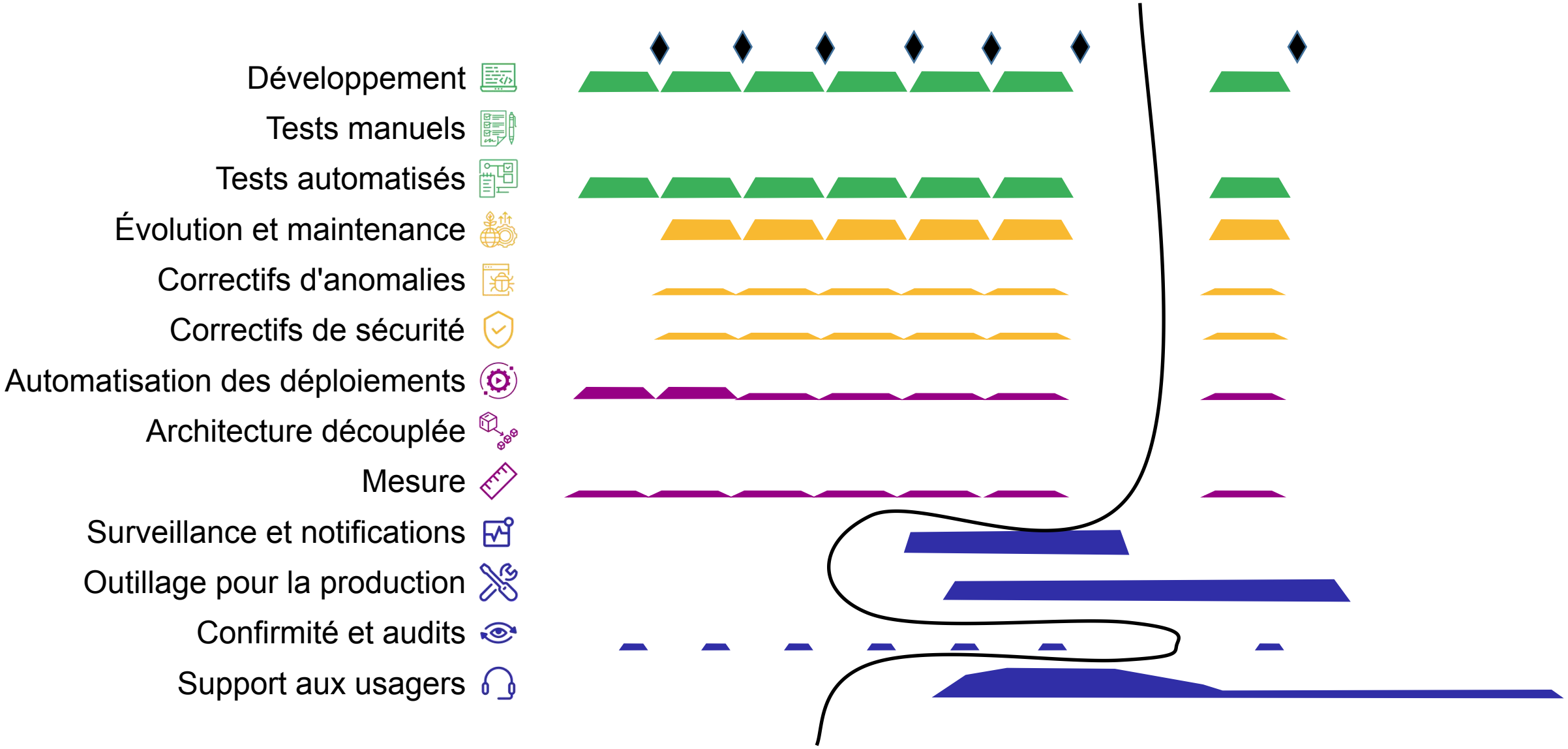


Les effets de la dette technique en Waterfall

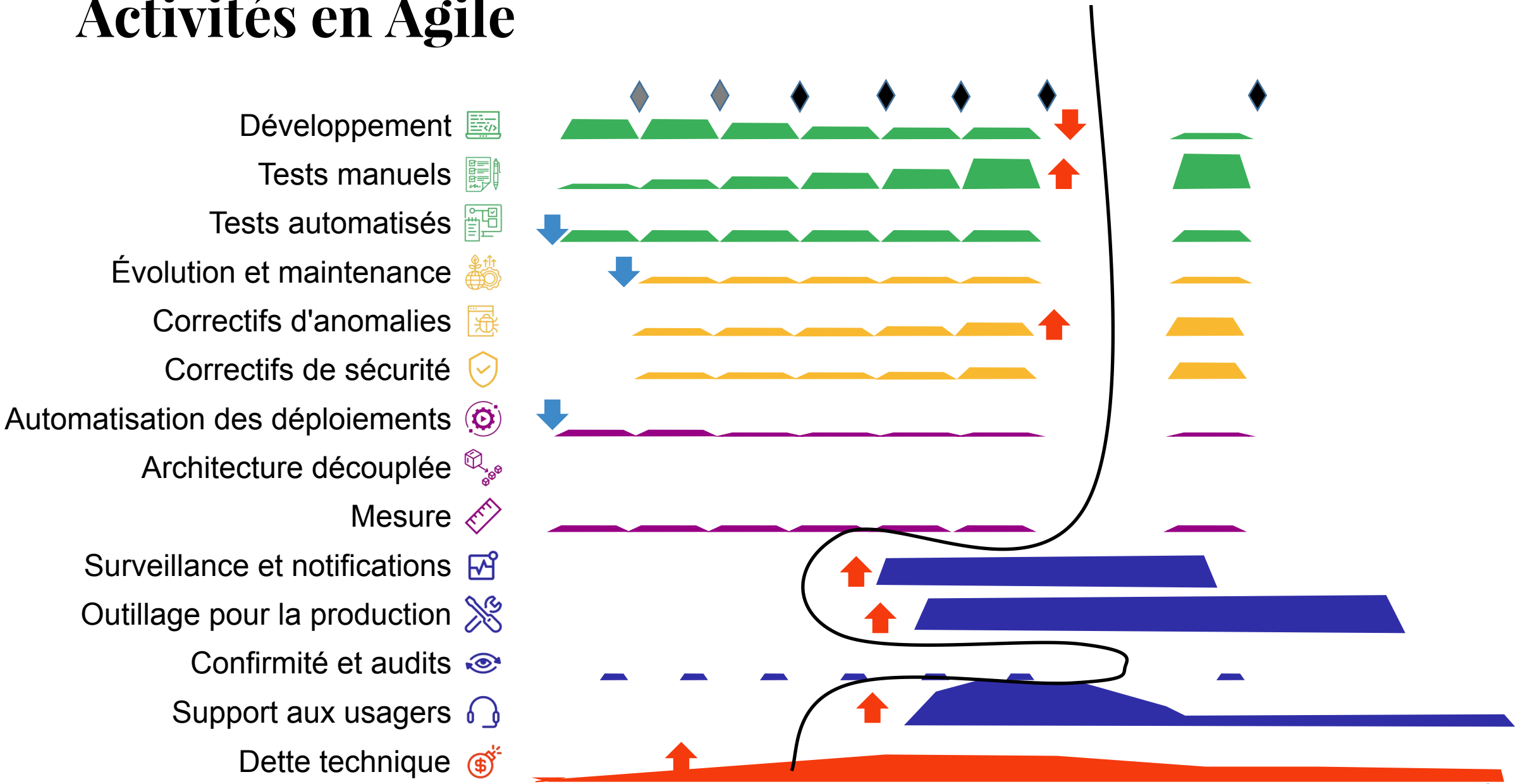
Waterfall → autre équipe qui va s'en occuper et vivre avec



Activités en Agile



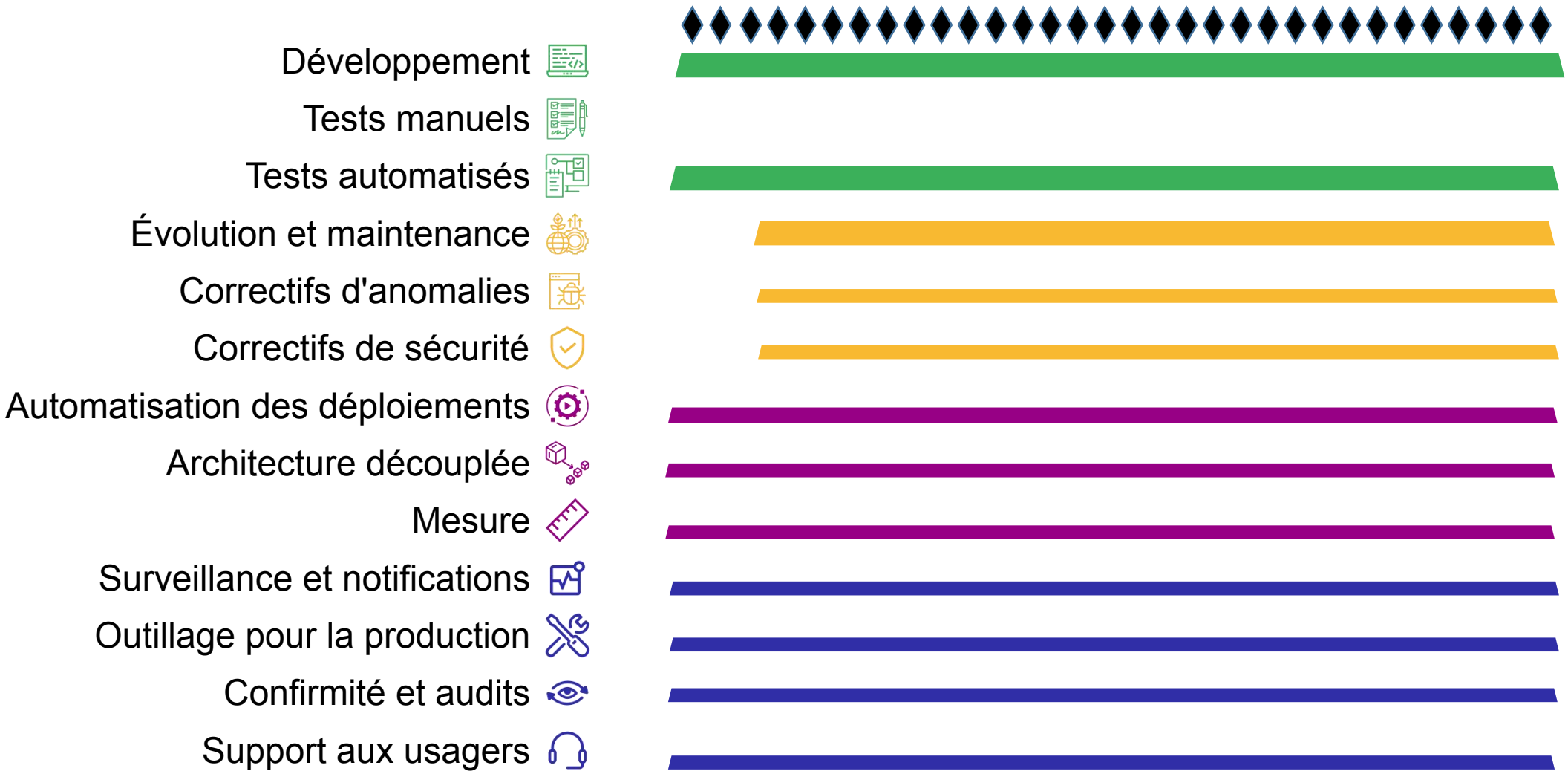
Activités en Agile



Les effets de la dette technique en Agile

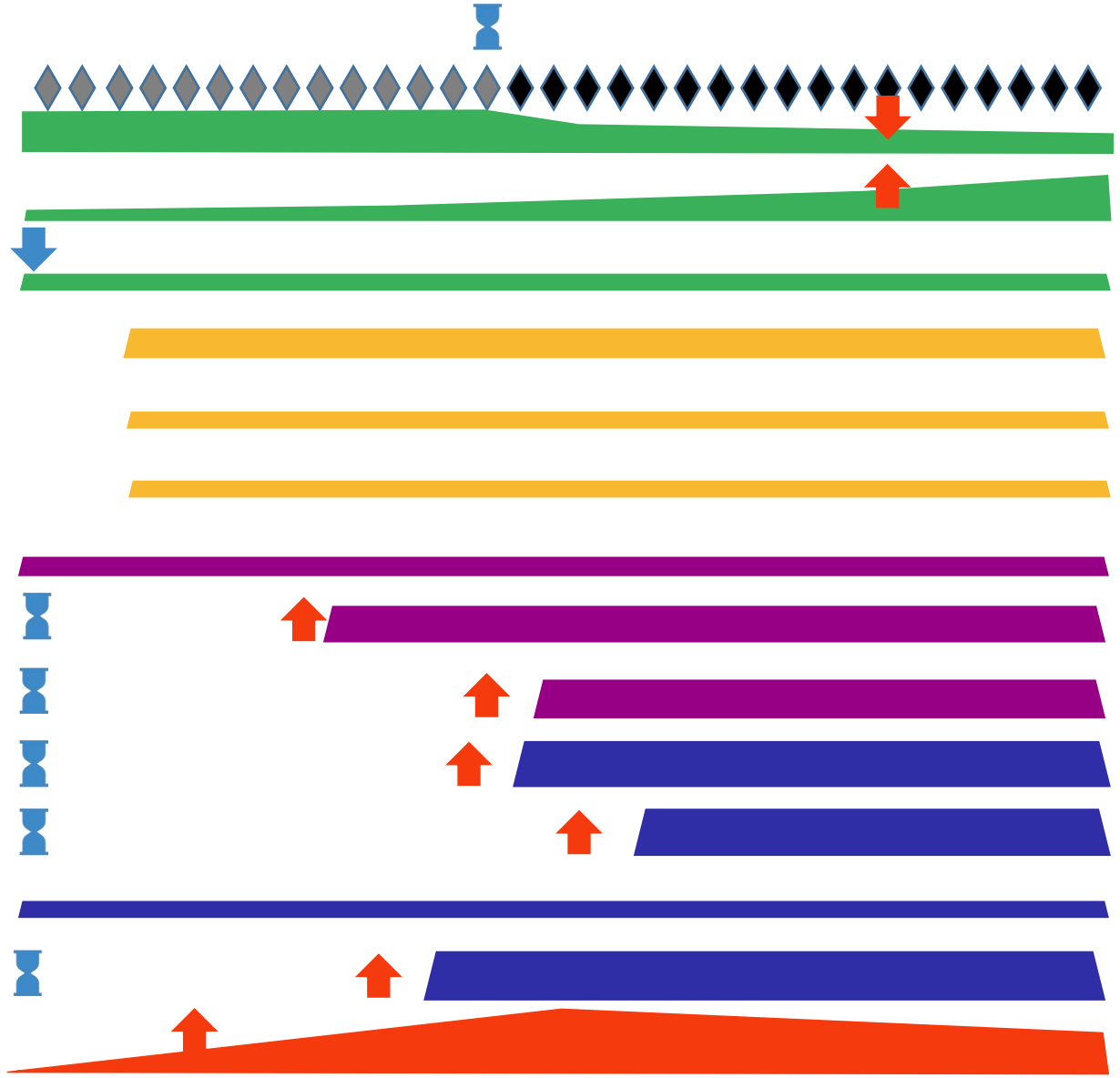
Agile → Ralentissement pour l'équipe pendant le développement, mais aussi pour les opérations

Activités en DevOps



Activités en Devop

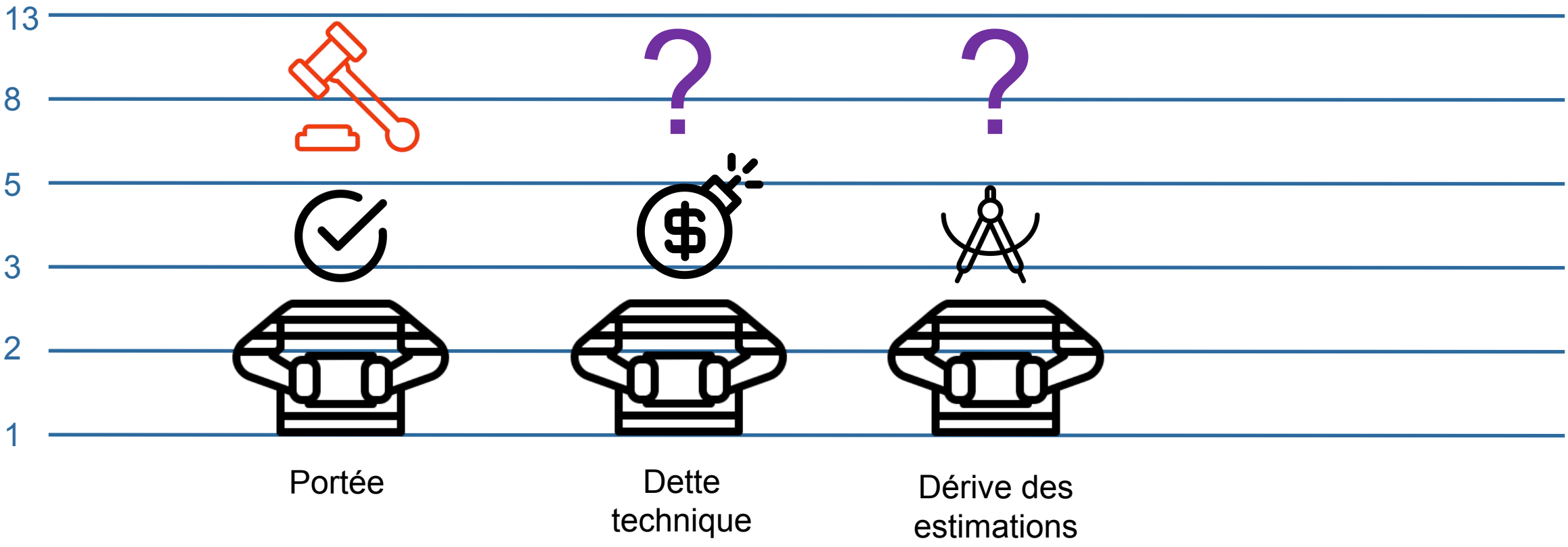
Développement
Tests manuels
Tests automatisés
Évolution et maintenance
Correctifs d'anomalies
Correctifs de sécurité
Automatisation des déploiements
Architecture découplée
Mesure
Surveillance et notifications
Outillage pour la production
Confirmité et audits
Support aux usagers
Dette technique



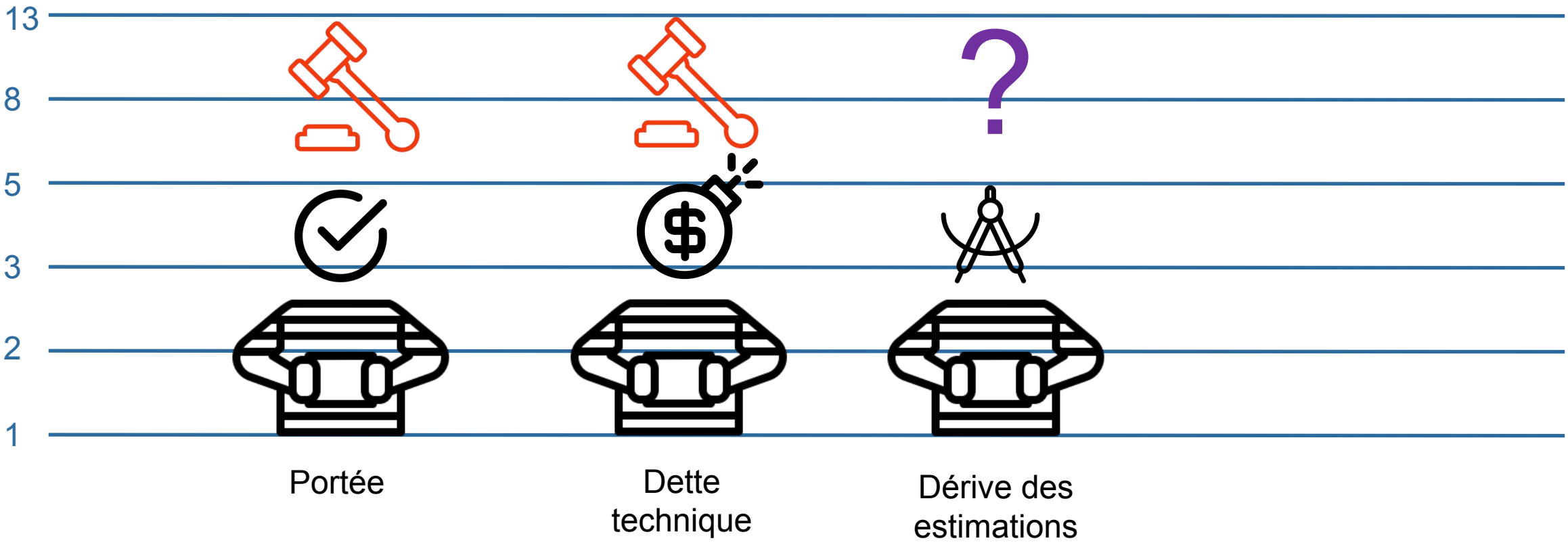
Les effets de la dette technique en DevOps

DevOps → Plusieurs sources de dette, incluant les opérations

La dette est-elle coupable ?



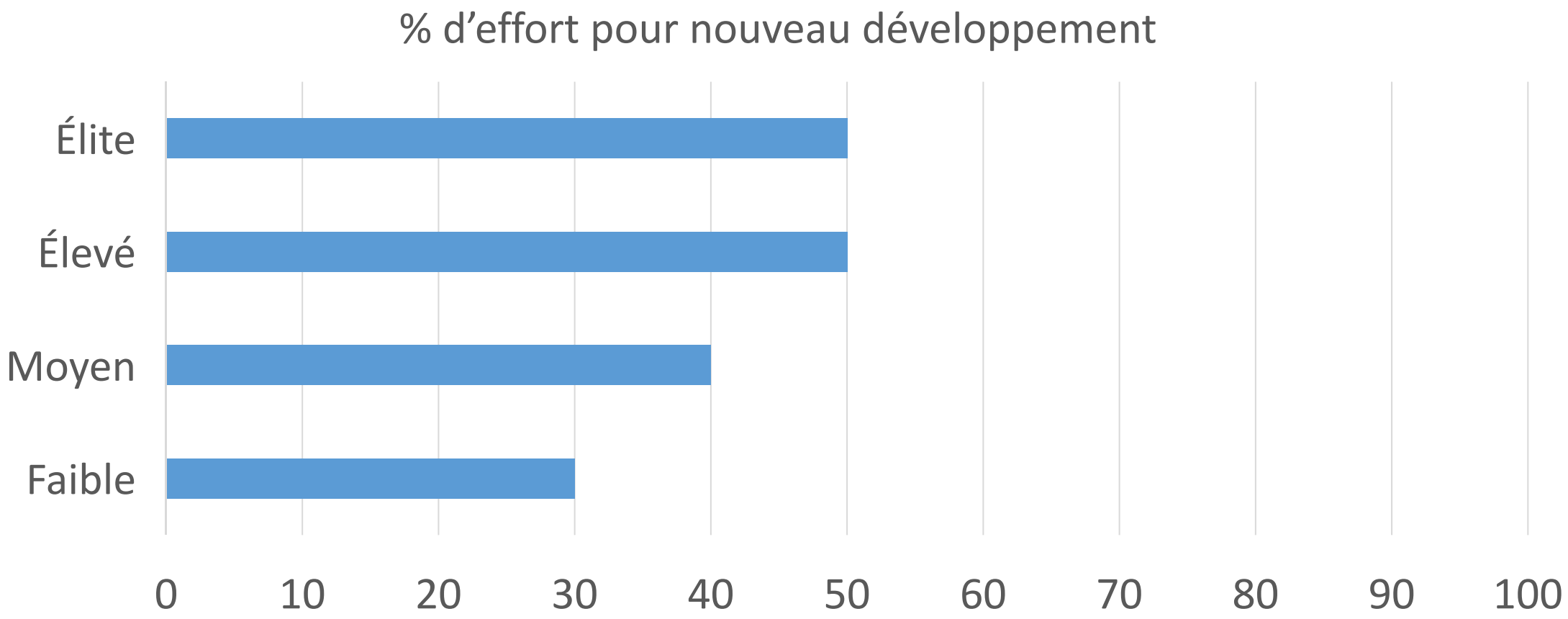
Coupables !



Les causes de la dette technique

- *Habitude de ne faire que des fonctionnalités*
  - *Accumule de la dette technique*
- *L'importance de la dette technique est sous-estimée*
- *En DevOps, beaucoup plus de travaux invisibles au PO et aux utilisateurs*

Médianes des ratios d'effort (State of DevOps, 2018)



Source: https://www.devops-research.com/research.html

Solution → pour mieux gérer la dette technique

- *Ajuster la définition de terminé pour éviter la dette technique*
- *Respecter de bons ratios:*
  - *Allouer assez de temps pour compléter correctement chaque récit*
  - *Aller du temps pour corriger les dettes techniques existantes*
- *Gérer la dette technique comme d'autres récits*

Un BON récit de Dette Technique, ça contient:

- *Cause*
- *Problématique*
- *Impact*
- *Solution*
- *Retour sur investissement*
- *Niveau d'effort requis*
- *Bénéfice attendu*

Comment choisir quelle dette régler ?

Considérer le type de dette et les coûts associés

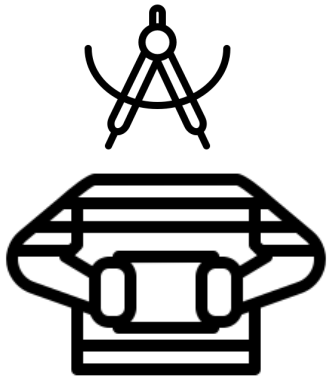
- *Frais récurrents → étapes manuelles à reprendre*
- *Intérêts composés → tout ajout va coûter plus cher par la suite*

Considérer le retour sur investissement

Considérer les objectifs d'affaires

Décrire adéquatement la dette technique
→ pour la rendre visible et compréhensible

Parlons de la dérive des estimations !



La dérive des estimations: un exemple

Niche 3 pts

Cabanon 5 pts

Bungalow 8 pts

Cottage 13 pts

Cottage 13 pts

Duplex 20 pts

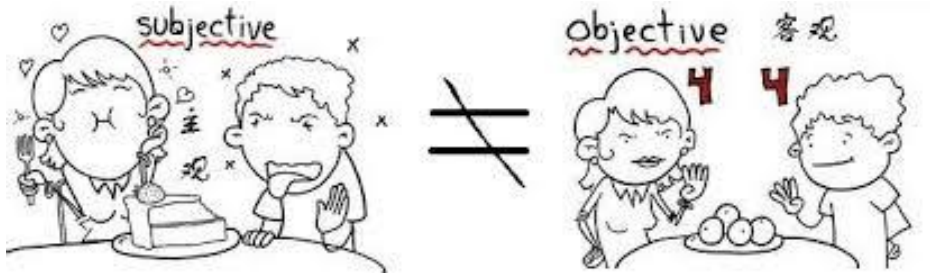
Bloc 40 pts

Cottage 8 pts

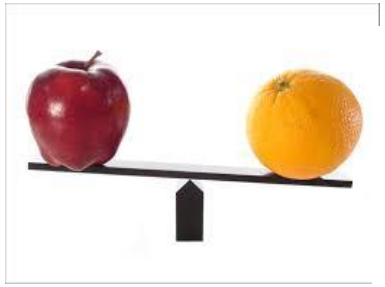
Cabanon 5 pts

Niche 2 pts

Le problème avec les USP…



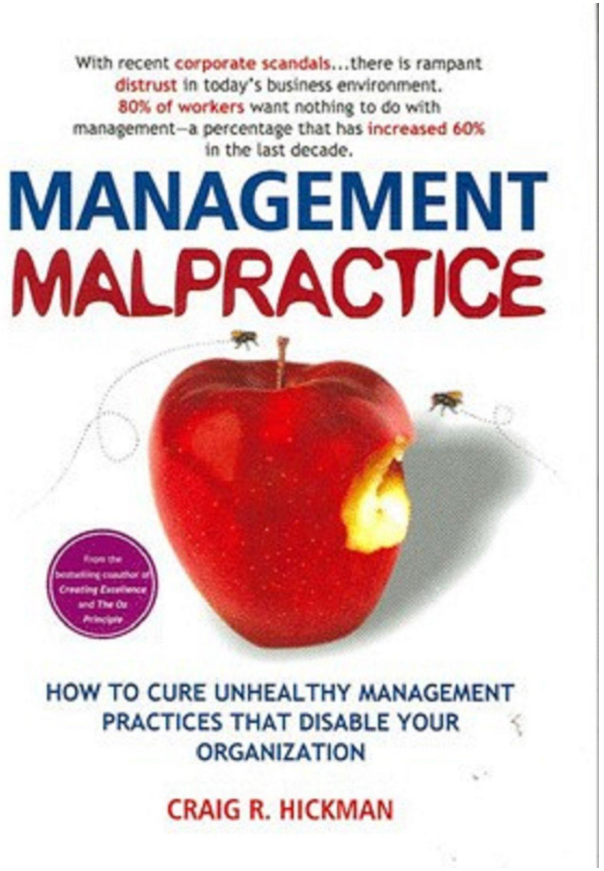
- *Valeur subjective*
- *N'a du sens que pour l'équipe dans CE projet:*
  - *Incomparable avec les autres équipes*
  - *Incomparable avec les autres projets de cette équipe*



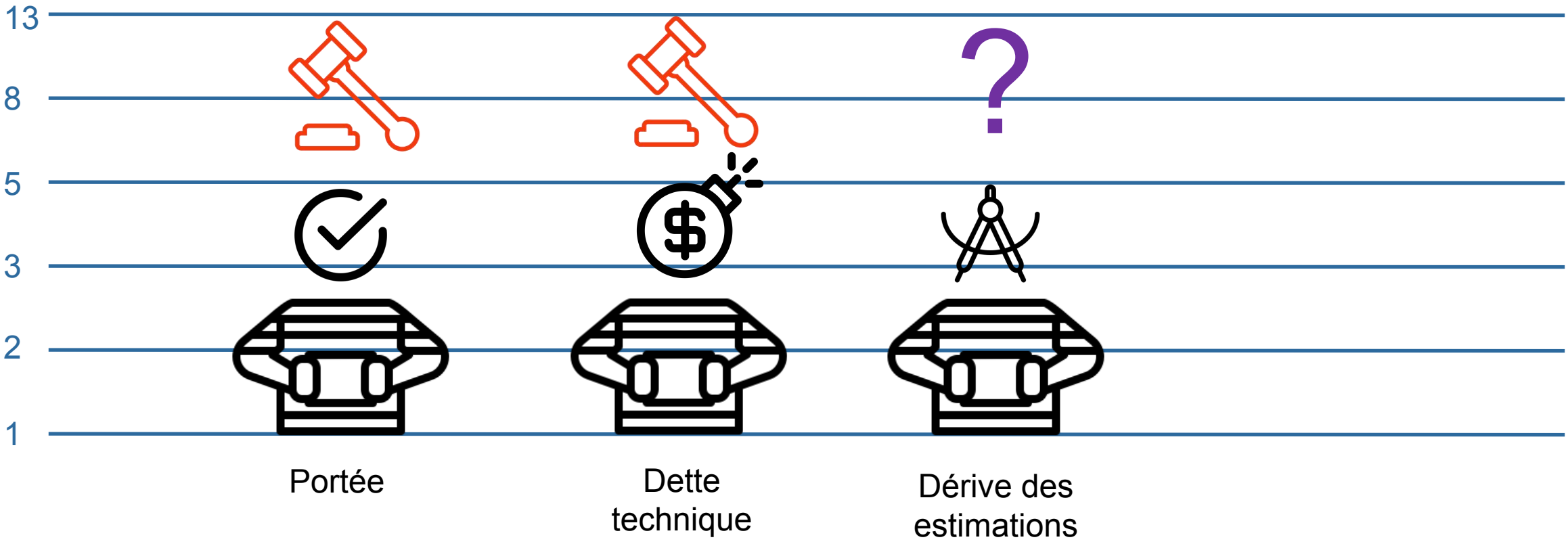
- *Alors… ce « niveau d'effort largement approximatif » est parfois (ou souvent) utilisé comme si c'était une mesure objective et comparable*

  →Et cette pratique est considérée comme de « faute professionnelle » en gestion!

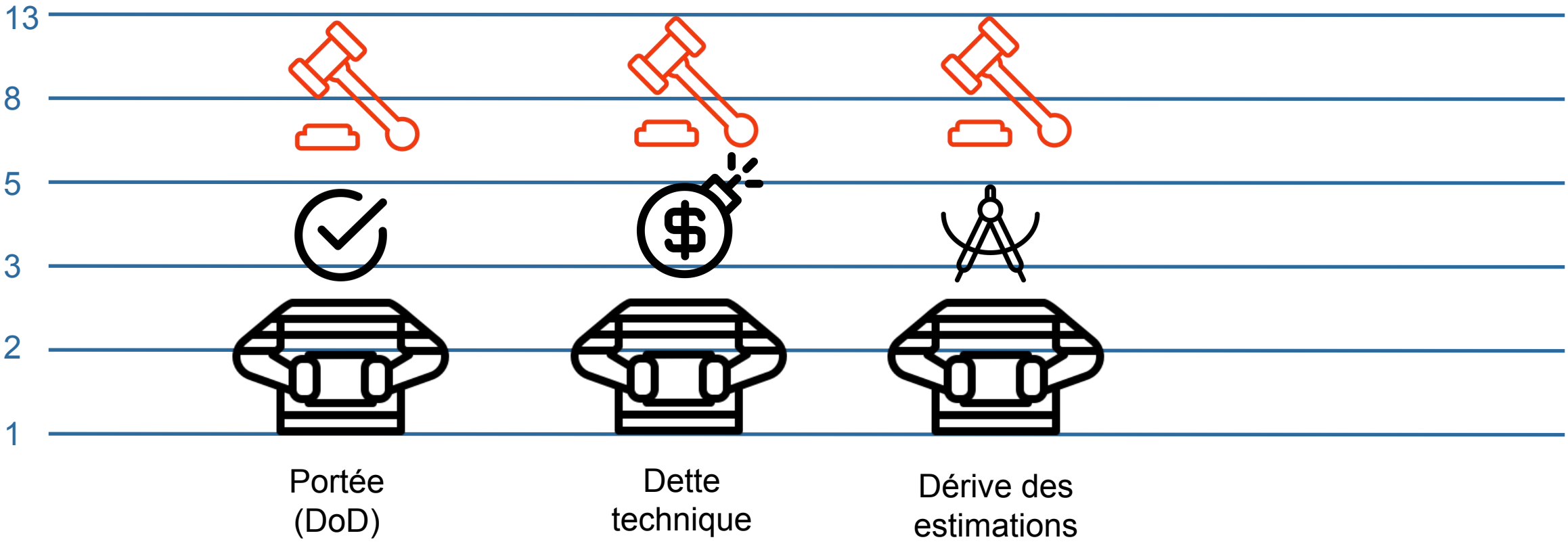
  →Immanquablement, on aura des écarts parfois important sur nos estimations!



La dérive des estimations est-elle coupable ?



Coupables !


13
8
5
3
2
1
Portée (DoD)
Dette technique
Dérive des estimations

Les causes de la dérive des estimations

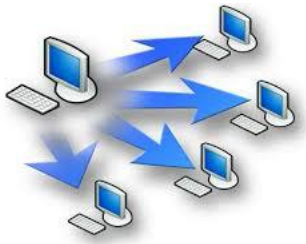
- *Les estimations ne sont pas «ancrées» sur une référence*
- *Tendance à sous-évaluer les travaux répétés*
- *Tendance à dire 1 USP = 1 jour*
- *Estimation != mesure*

Estimations découlant de nos données de vélocité

Productivité (ou vélocité)

Effort de l'équipe

Coût



Taux de livraison

Date

Problème: Comment mesurer la taille objectivement?

Taille du logiciel

Valeur d'Affaires

Source: Trudel, S. (2019). « Agile & COSMIC, a Good Integration! », CNMES, Mexico, 24 Octobre 2019.

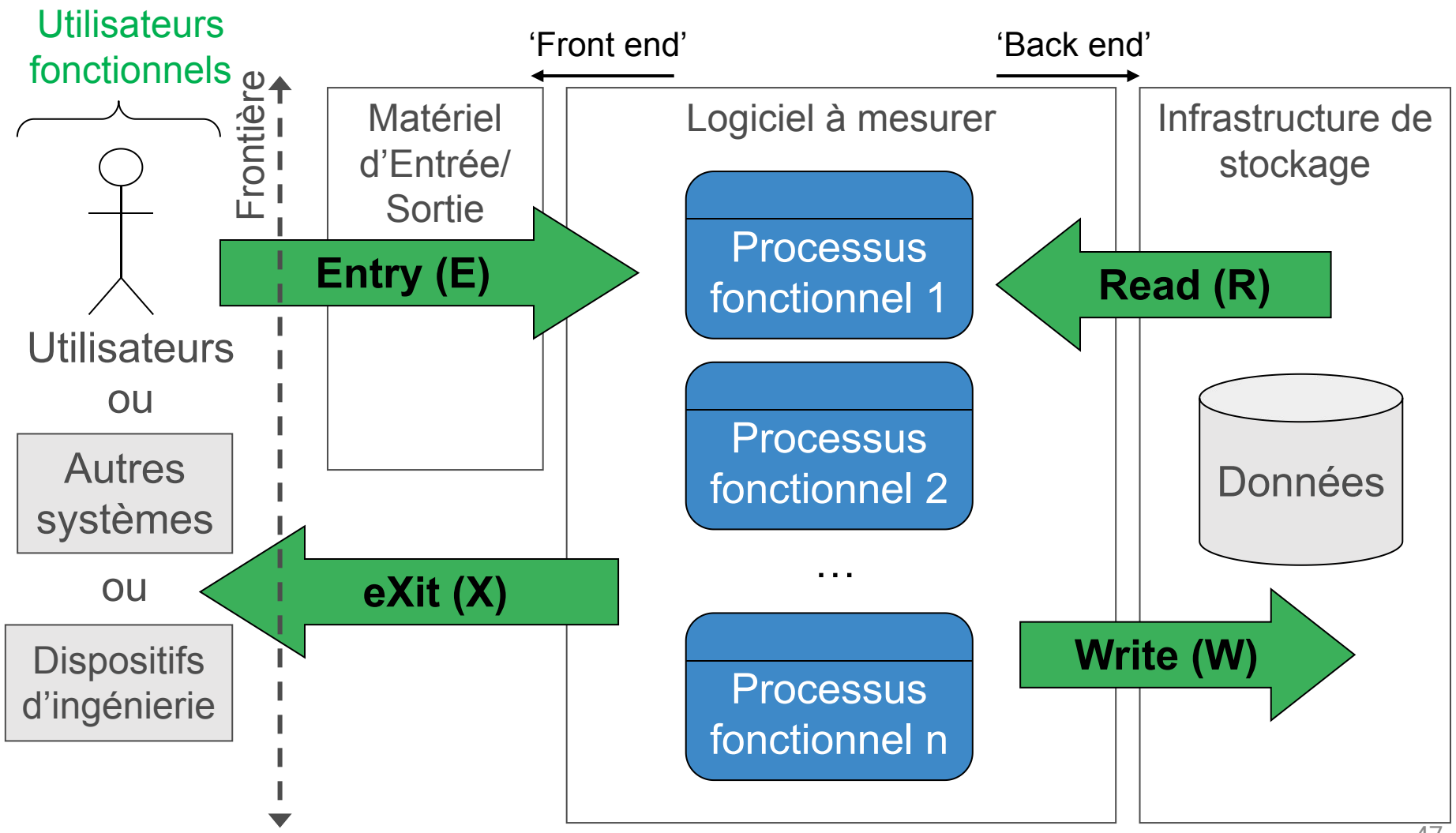
Solution → pour mieux estimer

- *Employer une échelle de référence*
- *Employer une mesure objective*

Pourrait-on utiliser l'analogie de la construction?

- *En construction: coût moyen = $/pi²*
- *En développement logiciel*
  = les données utilisées ou traitées:
  - *# Entrées, # Sorties, # Lecture, # Écritures?*
- *Mais OUI! Ça s'appelle la « méthode COSMIC »!*

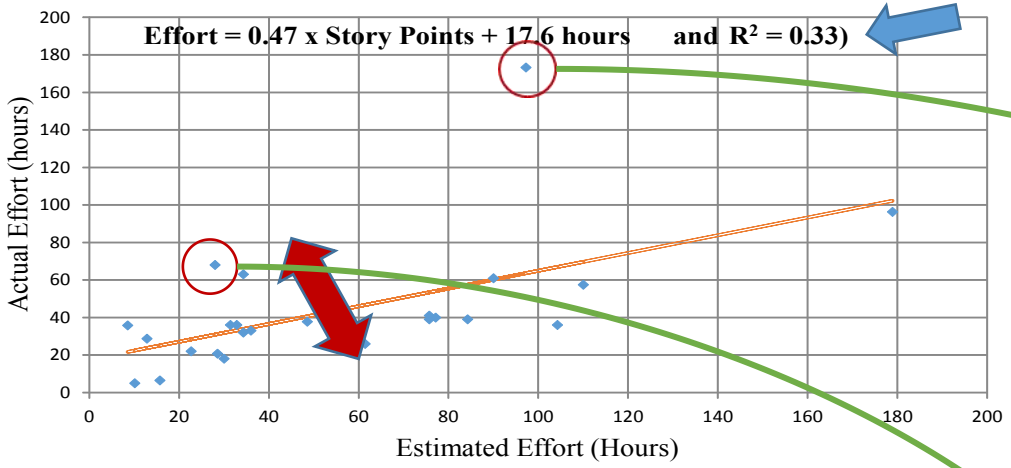
La méthode COSMIC (ISO 19761), en bref



Exemple de données réelles: logiciel de sécurité et surveillance

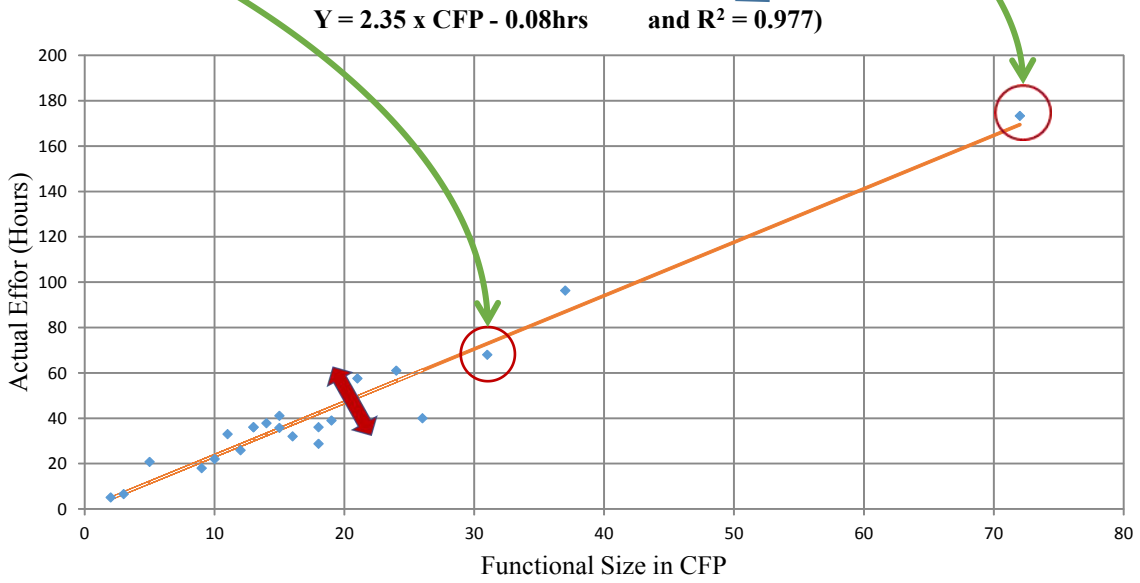
Scrum + TDD, sprints de 2 semaines:

- 24 tâches en 9 sprints
- Estimées en #USP
- Effort mesuré en #Heures
- Mesurées en #PFC



User Story Points

COSMIC

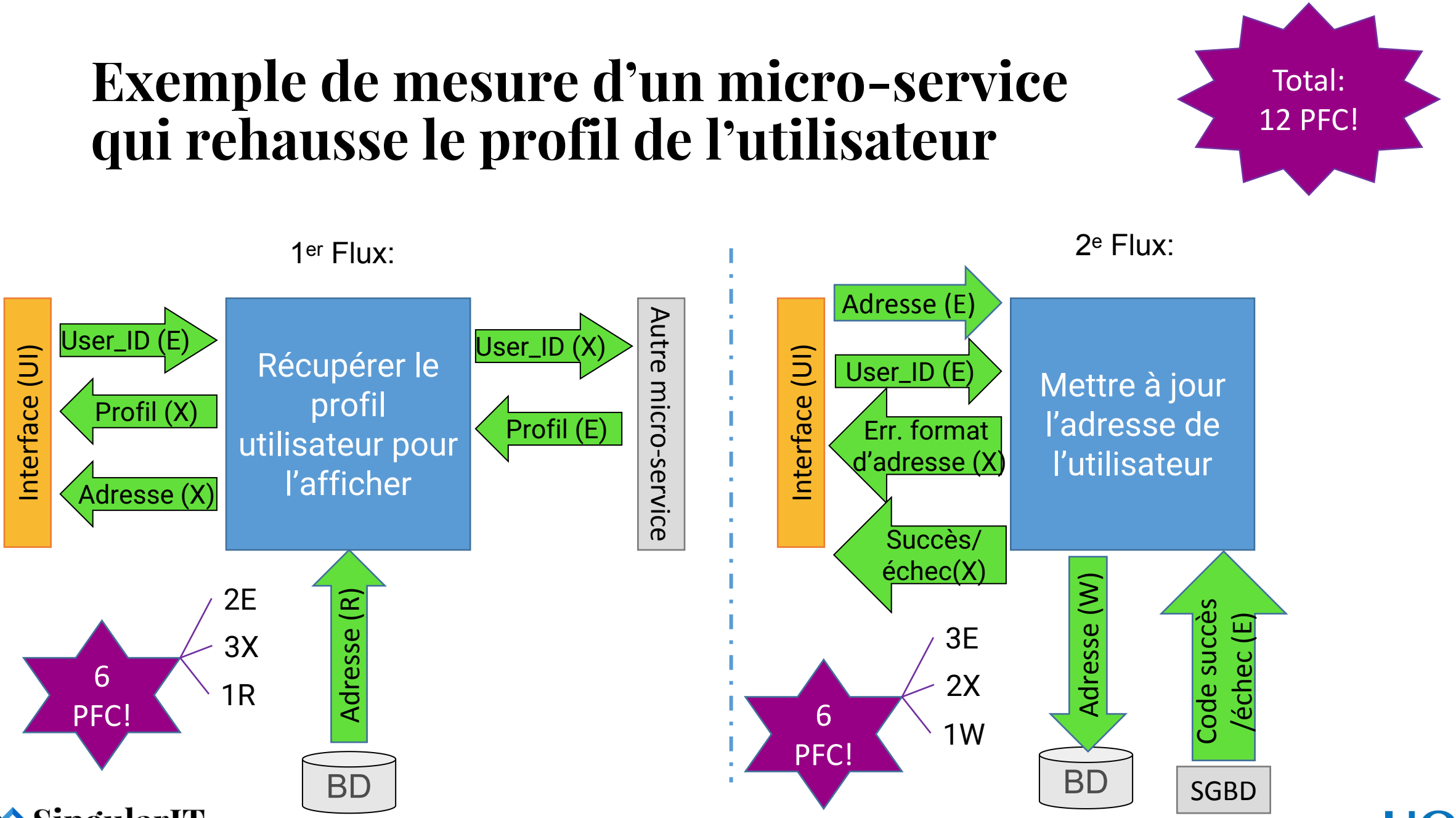


Source: *C. Commeyne, A. Abran, R. Djouab (2016).* «Effort Estimation with Story Points and COSMIC Function Points - An Industry Case Study». Disponible de *www.cosmic-sizing.org* 'Software Measurement News'. Vol 21, No. 1, 2016

SingularIT Solutions

UQÀM

Exemple de mesure d'un micro-service qui rehausse le profil de l'utilisateur



En DevOps, la vélocité ça pourrait être…

- *Coût unitaire: # Heures/PFC* → *coût de développement @200$/hr*

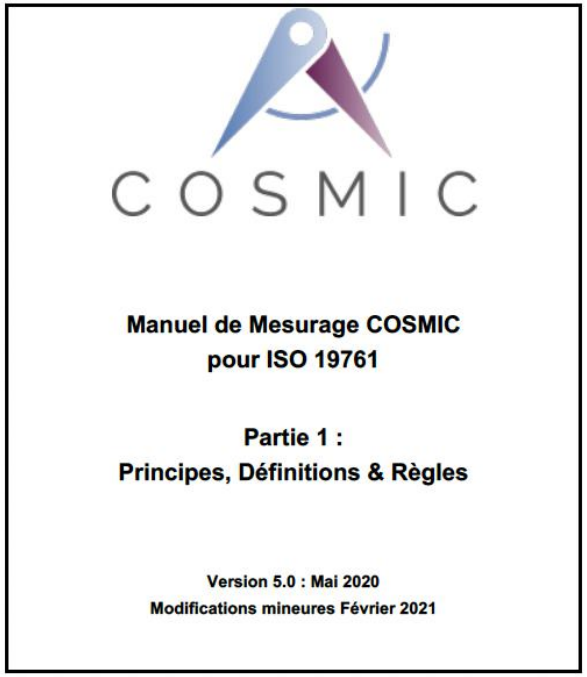
Incluant Dev, Test, Bug fix

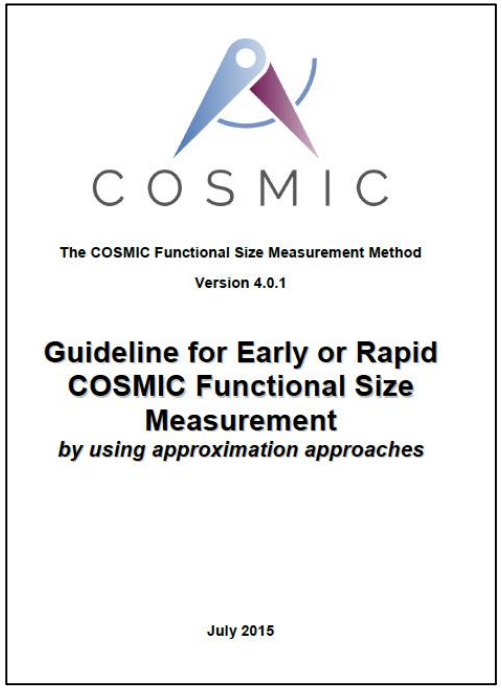
Mode	Vélocité	Effort	Coûts
Waterfall	10 à 45 Hrs/PFC	120 à 540 hrs	24 000$ à 108 000$
Agile	2 à 20 Hrs/PFC	24 à 240 hrs	4 800$ à 48 000$

- *Taux de livraison: #PFC/mois ou #PFC/sprint* → *date de livraison prévisible*

COSMIC c'est...

- *Une organisation sans but lucratif: https://cosmic-sizing.org/*
- *Mission: maintenir la méthode et les guides GRATUITEMENT*





Conclusion

- *On a identifié 3 coupables avec des solutions pour les contourner*
  - *Portée, avec plus de tâches*
  - *Dette technique*
  - *Dérive des estimations*
- *Encore faut-il appliquer les solutions adéquatement!*

Questions

Préenregistrement et diapositives (disponibles dans les 24h)

Diapositives DevOps et Vélocité

https://is.gd/L2cc00

Préenregistrement vidéo

https://is.gd/4poXhn

COSMIC

https://cosmic-sizing.org/