DÉCRYPTAGÉO LES RENCONTRES

# Agilité et qualité logicielle: une mutation en marche

Jean-Paul SUBRA

SoftMethods

# Introduction : le manifeste Agile

## Manifeste pour le développement Agile de logiciels

*Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire.*
*Ces expériences nous ont amenés à valoriser :*

*Les **individus et leurs interactions** plus que les processus et les outils*
*Des **logiciels opérationnels** plus qu'une documentation exhaustive*
*La **collaboration avec les clients** plus que la négociation contractuelle*
*L'**adaptation au changement** plus que le suivi d'un plan*

*Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers.*

Ce manifeste induit 12 principes sous-jacents, examinons quelques uns d'entre eux…

# Zoom sur quelques principes…

Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.

Accueillez positivement les changements de besoins, même tard dans le projets.

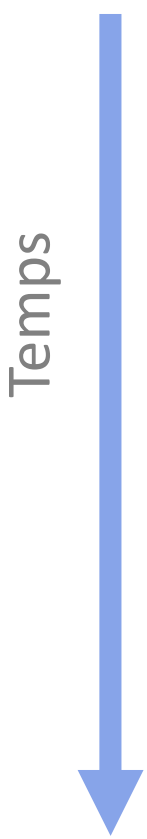
Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.

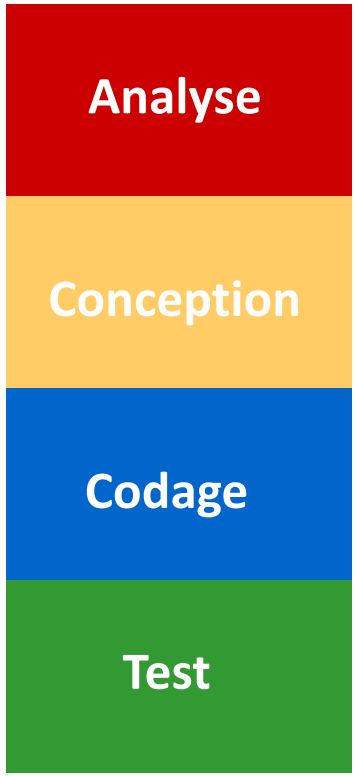
Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.

Un logiciel opérationnel est la principale mesure d'avancement.

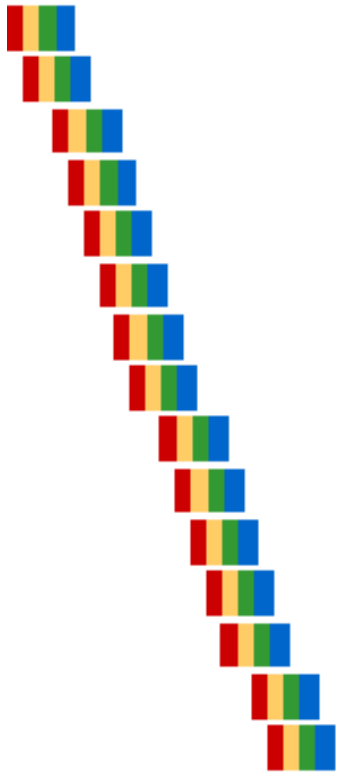
La qualité est présente partout dans les valeurs fondamentales de l'agilité

# Approches traditionnelles et Agiles : la différence en une image




Méthodes traditionnelles
(cycle en V…)



Agile

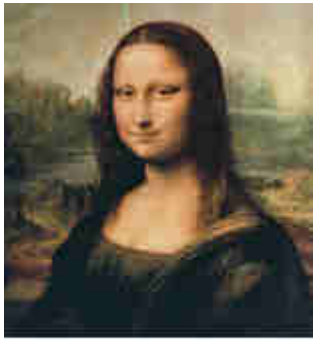
# L'approche traditionnelle: illustration

Ce qu'attend le client final…

**Specifications détaillées:**
une jeune femme énigmatique, aux cheveux longs, devant un paysage, etc….

Livraison au client de la Version 1

**6 mois**



Livraison au client de la Version 2

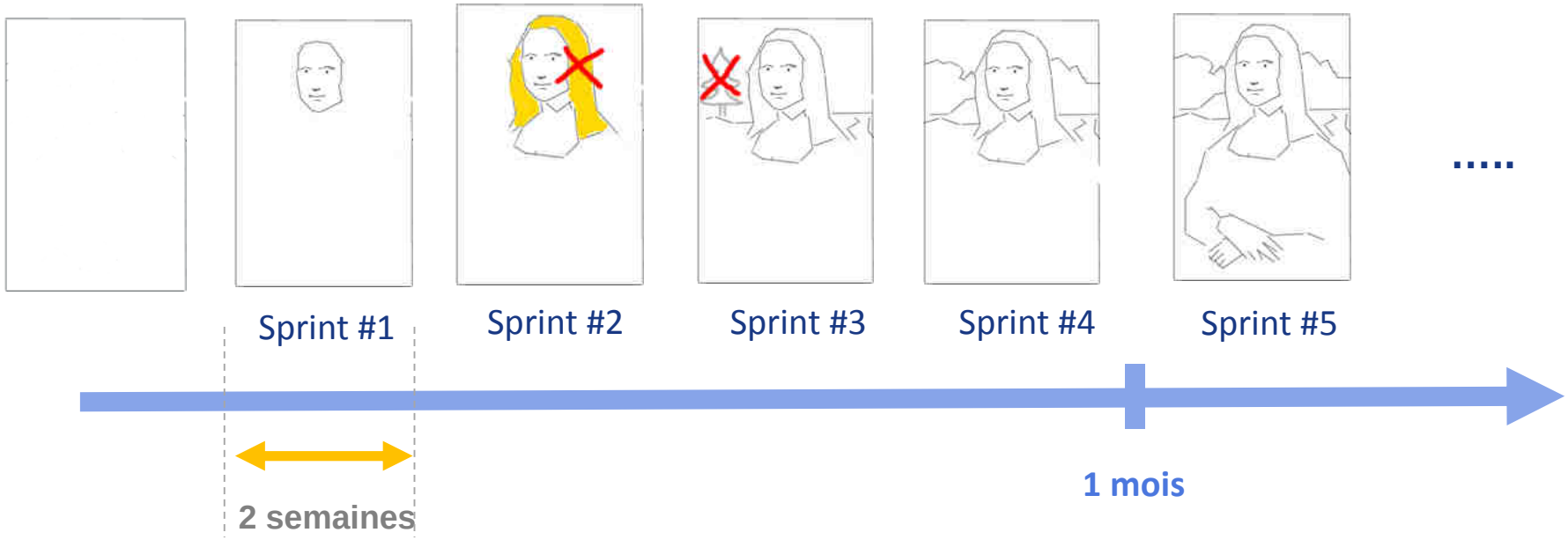
**12 mois**

Livraison au client de la Version 3

**12 mois**

Mais rien n'empêche qu'on arrive à ça…
… 6 mois de travail perdus

# L’approche agile




A la fin de chaque itération, on a un produit potentiellement livrable au client final (donc validé)

On bénéficie d’une boucle de réaction très courte pour traiter les problèmes de non-qualité (qu’il s’agisse de bugs ou d’inadaptation au besoin réel)

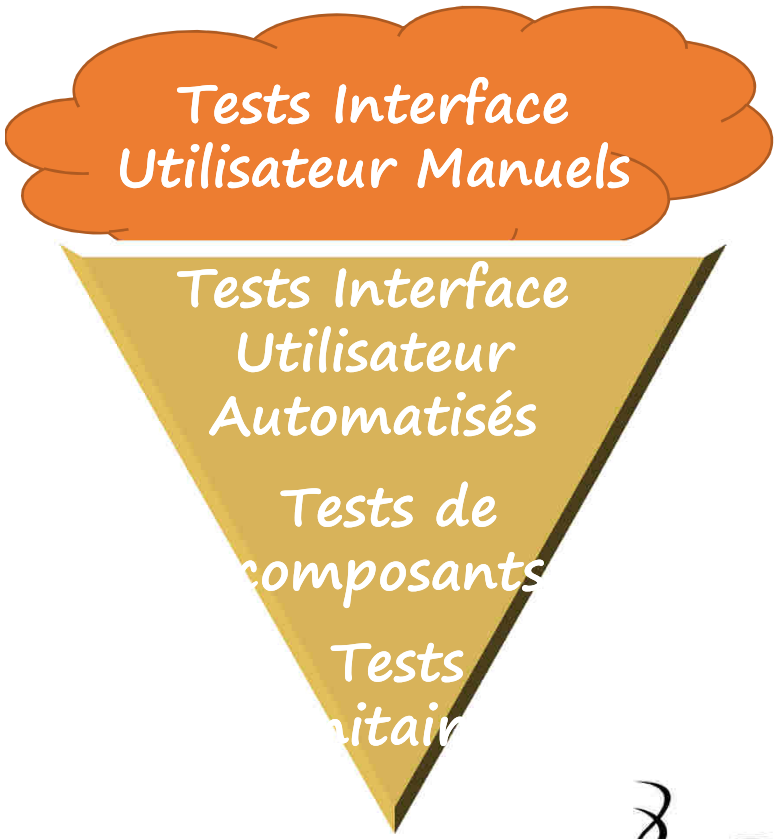
# Conséquence : des contraintes fortes sur la mise en œuvre de la qualité (1/2)

- On doit tester souvent
  - Intégration continue....et de plus en plus souvent déploiement continu (on livre tous les jours !)
- Puisqu'on doit tester souvent, on doit tester rapidement
- On doit répéter sans cesse des jeux de tests (tests de non régression) qui grossissent avec le logiciel
- Pour ce faire, l'automatisation est un must (mais elle n'est pas applicable à tout…)

# Conséquence : des contraintes fortes sur la mise en œuvre de la qualité (2/2)

- On doit tester le plus tôt possible dans le cycle:
  - l'approche qualité doit commencer dès le développement (TU : tests unitaires…) => le bon développeur agile est aussi un testeur !
  - les méthodes où on crée les tests avant de coder sont particulièrement efficaces (TDD: test driven development, ATDD: acceptance test driven development,…), mais encore insuffisamment pratiquées

# Anti-pattern : le « Cornet de glace »

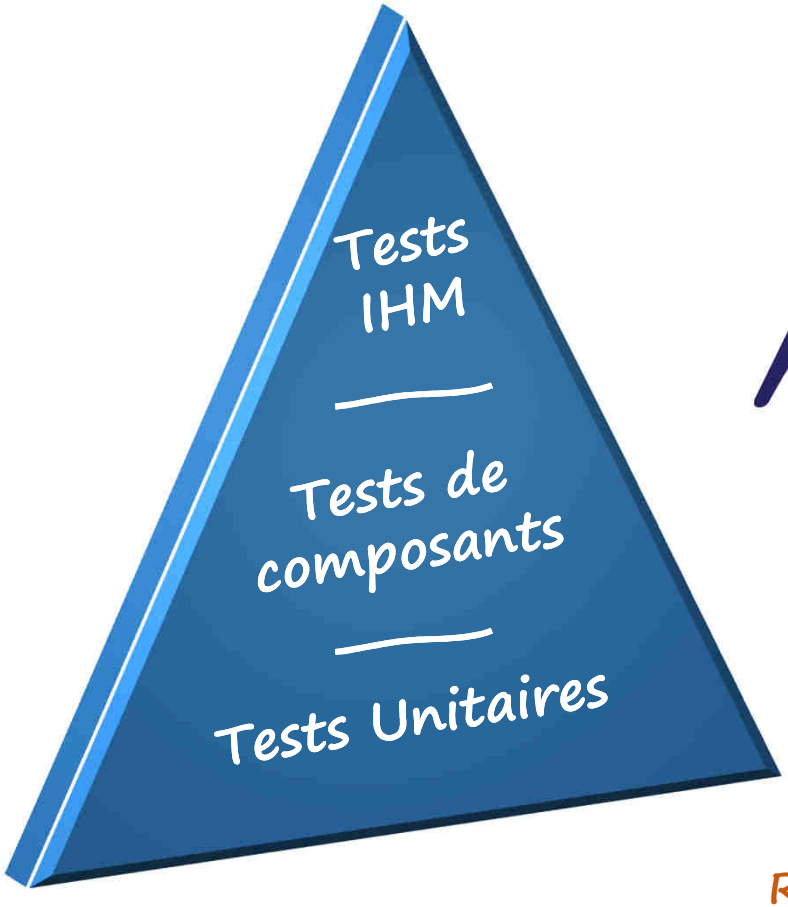


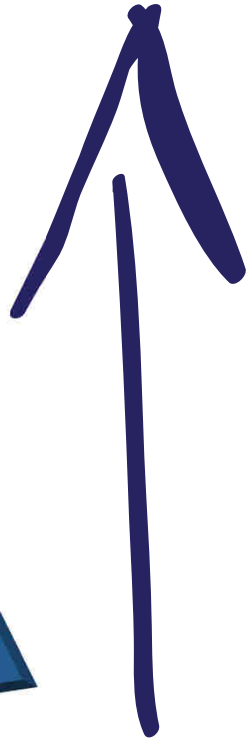

Encore souvent, le test du logiciel est fait principalement via son interface utilisateur (car intuitivement, c'est la méthode qui semble la plus simple)

A éviter en approche Agile (encore + que dans les méthodes traditionnelles), car : couteux à développer (et à automatiser), longs à mettre en place et à exécuter, peu réactif, peu précis => pour le test fonctionnel d'acceptance, privilégier le test à un niveau composant/API (mais il faut que l'application soit bien architecturée !)

**Analogie pour mieux comprendre: pour un véhicule qui sort d'usine, se rendre compte que le klaxon ne fonctionne pas en faisant un parcours de test sur route de 20 km n'est pas du tout optimal…**

# La pyramide des tests idéale

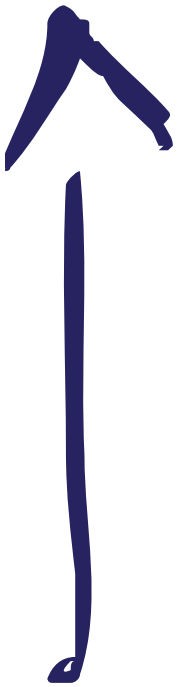


Long

Rapide

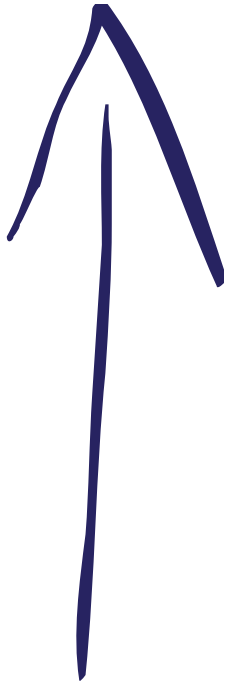
Temps d'exécution



Cher

Peu cher

Coût de création et maintenance



Faible

Forte

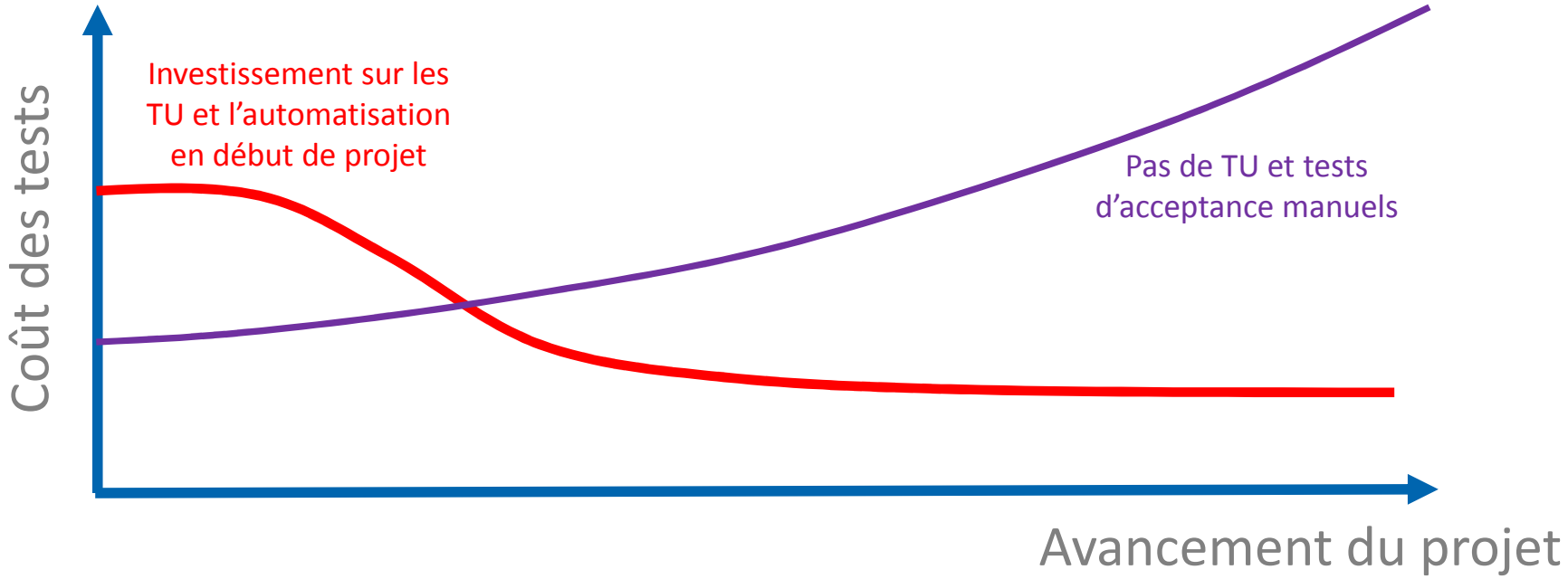
Précision, fiabilité

# Dans le monde Agile, la qualité c'est l'affaire de tous…

## … et donc c'est celle des développeurs

Discours souvent entendu sur les pratiques TU/TDD : c'est compliqué, ça coute cher à mettre en place.

Mais l'expérience montre que ça peut rapport gros…

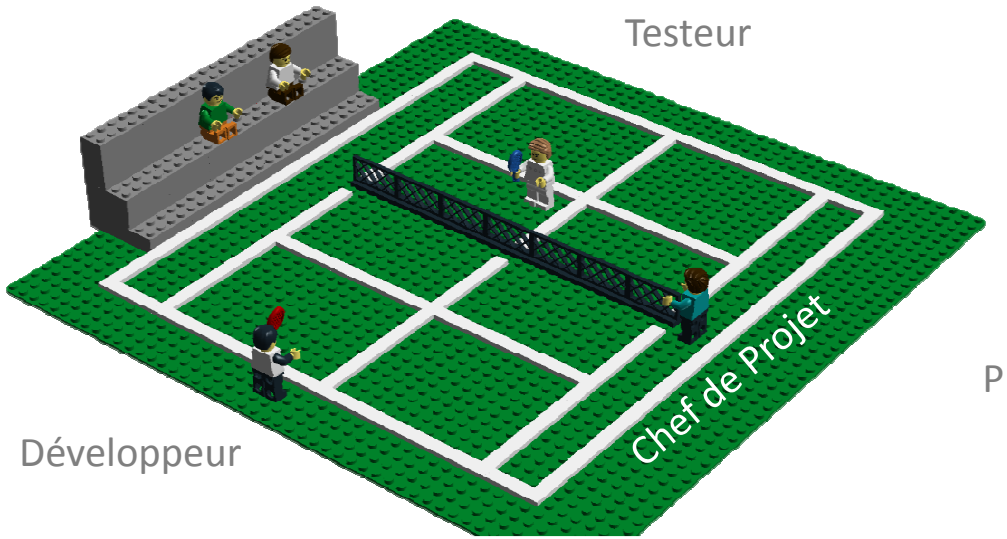


# Mais alors vous ne testez pas le logiciel comme ses utilisateurs ?

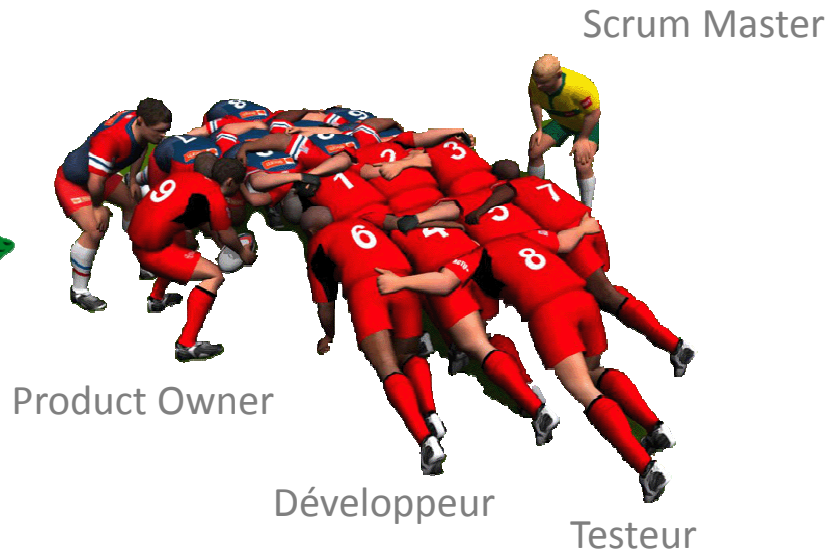
- En effet, mais on va plus loin…
- La stratégie qualité idéale dans un modèle Agile est une combinaison des différentes pratiques de test:
  - Beaucoup de TU automatisés
  - Un nombre important de tests fonctionnels à un niveau API/composants
  - Des tests d'interface utilisateur automatisés (attention, le but n'est pas là de tester le fonctionnel, mais le bon fonctionnement de l'interaction avec l'utilisateur)
  - Quelques tests « end to end » (scénarii complets d'utilisation), largement manuels
  - Ne pas oublier tout le reste (tests de performance, montée en charge, sécurité…)

# Les testeurs et les autres…

## Avant…




## En mode Agile…




Par construction, les testeurs font intégralement partie de l'équipe Agile: les isoler des autres comme on avait parfois l'habitude de le faire, est une erreur fatale.

La collaboration est permanente (avec les développeurs et les représentants du client final).

# Testeur Agile: un métier en mutation

- Etre testeur est un métier valorisé dans l'environnement Agile => le testeur est un élément central du processus de développement Agile
  - Ce n'est plus une punition ou un choix par défaut…
  - On ne s'improvise pas testeur: nécessite connaissances méthodologiques et techniques, capacité à comprendre le fonctionnel, bonne communication.
- Avec la nécessité d'automatiser, on a besoin, en plus des profils traditionnels, de testeurs qui savent aussi développer (par ex, des scripts de test, mais parfois aussi aller dans le code du logiciel qu'ils testent)
  - « Software Engineer in Test » dans l'approche Google
  - Test et développement sont plus proches que jamais: certains sociétés ne distinguent plus les 2 (pas de testeurs !)

# Entendu récemment: « L'Agile, c'est pas bien…

…on n'arrive pas à tester car rien n'est terminé à la fin des sprints »

Dire qu'on a mis en place une démarche Agile, ça ne suffit pas et quand bien même aujourd'hui presque tout le monde le prétend* il faut le faire en comprenant les fondements, sinon ça ne marche pas et la qualité ne sera évidemment pas au rendez-vous.

Passer à l'agilité, nécessite d'abord un réel investissement en formation et en accompagnement car c'est un changement majeur par rapport au modèle selon lequel la plupart des développeurs ou testeurs ont été formés.

Il y a des outils très simples permettant de s'assurer que les fondements sont connus et maîtrisés, comme les Professional Scrum Assesments de Scrum.org.

Une fois que le déploiement est terminé, et bien, ce n'est pas terminé... car il faut en permanence évaluer ses pratiques et les adapter.

* le World Quality Report indique que 93% des entreprises ayant répondu en 2014 ont conduit au moins un projet agile

# Conclusion

- Le modèle Agile intègre la qualité au cœur du processus de développement, de manière continue au cours des projets logiciels
- Pour en tirer les bénéfices, il est indispensable de mettre en place les pratiques adaptées: TU (et idéalement TDD), automatisation des tests, intégration continue,…
- Bien appliqué (ce qui nécessite un investissement en formation et une dynamique d'amélioration continue), il permet de délivrer de meilleurs logiciels et de limiter les gaspillages (grâce aux cycles courts)
- Il induit une transformation/une professionnalisation du rôle des testeurs, qui voient (enfin) l'importance de leur mission complètement reconnue

# Merci !

SoftMethods

www.softmethods.fr

jean-paul.subra@softmethods.fr