# Les enjeux de l'intégration continue

Cédric TESNIERE & Maxime HORCHOLLE

January 19, 2013

## Contents

# 1 Introduction

On entend de plus en plus parler d'intégration continue dans les médias spécialisés, de plus en plus d'entreprises se spécialisent dans ce domaine, mais le public sait-il réellement de quoi il s'agit. Sans doute, sait-il que cette technique sert à améliorer la qualité du code, mais en sait-il réellement plus. C'est pour cela que nous avons décidé dans ce document de faire un panorama très exhaustif de ce qu'est l'intégration continue.

Pour répondre à notre sujet ce document sera découpé en trois grandes parties. Une première expliquera en quoi consiste globalement l'intégration continue et ce qu'elle peut apporter à un projet. Dans la seconde nous verrons les types d'outils les plus utilisés et pourquoi les mettre en place. Et enfin, dans notre dernière partie nous aborderont les outils encore peu connus et très peu utilisés par les entreprises qui pourraient bien révolutionner ce qui existe à l'heure actuelle en matière d'intégration continuent et pour cela nous irons regarder ce qu'il se passe du côté du monde de l'open-source.

# 2 L'intégration continue

## 2.1 Principe

## 2.2 Les motivations des entreprises

La question d'utilisée l'intégration continue se pose avant que le projet commence de la même façon que les entreprises, donc pourquoi de nos jours les entreprises optent pour l'intégration continue ? La question est simple mais reste complexe dans le cas où certaine entreprise reste perplexe à cette pratique car ils ne l'ont généralement jamais testé.

Marketing

- Demande de démonstrations non planifiées

Budgets

- Démontrer rapidement l'avancement d'un projet
- Projets gérés par tranches, par lots conditionnels : focus sur le fonctionnel important !

Ressources, équipes

- Coordination d'équipes distribuées : le reporting projet ne suffit pas !
- Il faut partager les mêmes éléments d'évaluation de l'état d'avancement d'un projet
- Des changements dans l'organisation : fusion/acquisition, restructuration, ...

Besoins : les besoins varient continuellement en fonction

- Des produits de concurrents éventuels
- Des changements légaux, règlementaires (contraintes d'importation, deconfidentialité, etc.)

Besoin d'intégrer les évolutions d'un projet en continu

## 2.3 Les motivation au niveau projet

Nécessité d'améliorer :

- La qualité des livrables
- Réduire la complexité (meilleure maintenabilité)
- Adéquation
- La traçabilité
- des changements
- des déploiements
- La productivité
- Se focaliser sur le métier, pas sur la technique

Principes « agiles »

- Fabriquer souvent•Tester souvent (tests unitaires)
- Tester les performances souvent
- Intégrer souvent dans le SI

### 2.4 Qu'est-ce que c'est et pourquoi l'utiliser

# 3 Les outils les plus utilisé du marché

## 3.1 Tests

### 3.1.1 Les tests unitaires

### 3.1.2 Les tests fonctionnels

### 3.1.3 Les tests d'intégrations

## 3.2 Gestionnaire de versions

## 3.3 Détecteur de copier coller

Rien qu'au titre on voit tout de suite l'intérêt de cet outil en effet il permet de verifier que le code n'est pas remplis de copier/coller plus ou moins justifié. En effet la multiplication des copier/coller rend le code de moins en moins maintenable car si un morceau de code copier/coller un peu partout dans le programme est modifié à un endroit car il entraîne des bugs il faudra mettre aussi à jours tout les autres morceaux où le code à été copier/coller. Ce qui rends donc le code très vite inmaintenable.

L'outil lui va aider les équipes à détecter les morceaux de code récopié se qui permettra donc aux développeurs d'identifier les parties du code qu'il faut factoriser au plus vite pour rendre le code plus maintenable et éviter les modifications en chaine de code, les bugs et régressions.

Exemple de détecteur de copier/coller:

Mess Detector http://fr.wikipedia.org/wiki/PMD_(logiciel)

## 3.4 Revue de code

## 3.5 Analyseur de code

## 3.6 Logiciel de suivi de problemes

Aussi communément appelé Bugtrackers se sont tout les outils qui servent à suivre l'évolution du projet. Dedans sont consigné les évolutions et les bugs de l'application. Ce qui est très utile pour gardé un historique de tout ce qui à été réalisé jusqu'alors. C'est aussi un outils indispensable pour distribué les tâches.

Avant la création de ce genre d'outils la gestion de projet était bien plus difficile et l'équipe était beaucoup moins réactive puisque que l'information était souvent soit non mise à jours ou pire complément indisponible. Avec un bugtracker efficace c'est un jeu d'enfant de rentrer des tâches, des bugs, de voir l'état d'avancement de ses collègues. Les bugs trackers moderne propose aussi de nombreuses autres fonctionnalités comme des forums, des diagrammes de gantt, etc...

La majorité des bugs tracker peuvent/dovient être installé sur un de ses serveurs dans ce genre de solutions on trouve Redmine (gratuit) ou Jira (payant pour les projet non open-source) qui sont de très bon bugs trackers pour les gros projets . Mais récemment un nouveau type de bugs trackers est apparu: les bugs tracker sur le cloud, ils sont souvent fait pour être utilisé avec les méthodes agile et sont surtout là pour les projets de petites envergures (équipe restreinte, taille du projet limité). Ces solutions sous souvent gratuite de base et propose un plan payant pour les utilisateurs professionnels. Dans cette catégorie on trouve des site comme Trello ou Assana.

## 3.7 Test de couverture

## 3.8 Coding style checker

Cet outil fait partie des outils importants pour facilité la relecture et la maintenabilité du code. En effet Il permet de tester si les normes de codage fixé sont bien respecté par les programmeurs. Ces règles doivent être respecté car si elle ne sont pas suivit à la lettre on se retrouve en général avec un code non uniforme ce qui ne simplifie pas sa lecture.

Voici quelques règles assez communes:

- Utiliser des quatres espaces au lieux des tabulations (ce qui permet d'avoir le même code sur n'importe quelle machine).
- Une ligne ne doit pas faire plus d'un certains nombre de caractère. Le nombre de caractère est en général situé entre 80 et 120. Si une ligne fait plus en général il vaut mieux la découper sur plusieurs ligne se qui facilitera sa lecture.
- La façon écrire du code. Par exemple doit on sauter une ligne après la fin d'une méthode, doit on mettre un espace entre une le if et la parenthèse.

Il existe des règles de codage que l'on considère comme des références dans chaque langages qui sont en général décidé avec des membres important de la communauté. Pour illustrer prenons la norme PSR-2 de PHP[1]. Il existe aussi les conventions pour le langage java[2].

Maintenant pour en revenir à l'outil qui gère ces conventions, il a tout sont intérêt car si un développeur ne les respectent pas les erreurs seront tout de suite mises en évidence et il pourra

[1] https://github.com/php-fig/fig-standards/blob/master/accepted/fr/PSR-2-coding-style-guide.md
[2] http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-139411.html

les corriger rapidement se qui évitera que le code ne devienne illisible dans un futur proche, car un code homogène est un réel plus pour sa compréhension et sa maintenabilité.

Quelques exemple d'outils, les outils en général sont nommé CheckStyle dans chaque langage:

Checkstyle pour Java:https://github.com/checkstyle/checkstyle

PHPCheckStyle pour PHP: https://code.google.com/p/phpcheckstyle/

### 3.9 Serveur d'integration continue

## 4 Ça va bientôt arriver!

## 5 Conclusion