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.

D'un point de vue du Marketing, l'utilisation de l'intégration continue permet d'avoir des demandes de démonstrations non planifiées. Le projet étant constament compilé et envoyer sur le serveur de dév, cela permet au client de visualiser le rendu du projet a chaque build.

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

Un gestionnaire de versions sert à stocker, versionner et partager son code. On utilise surtout des gestionnaires de versions quand le besoin de partager le code source entre les différents membres d'une équipe se fait ressentir. En effet le plus souvent stocker son code sur un gestionnaire de source alors que l'on est le seul utilisateur à peu d'intérêt, à part de celui de ne jamais perdre ses sources. Les plus gros avantages se font ressentir sur les projets où il y a une équipe en effet voici une liste des gros avantages d'avoir un gestionnaire de sources quand on est en équipe :

- Tous les membres de l'équipe ont toujours une version du code à jour. Donc un gain de temps énorme lors de la mises à jour.
- Garder un historique de toutes les modifications et de qui les à effectuer. Ce qui permet un retour rapide en cas d'introduction de bug et de plus avoir peur de modifier le code source.
- Géré finement vos versions. Par exemple figer une version stable qui ne bougera plus et qui contient un nombre limité de bugs (une version "stable").

Voilà les avantages de tous les gestionnaires de sources il faut savoir qu'il existe deux types de gestionnaire de sources les centralisés qui existe depuis des années et les décentralisés qui inondent le marché depuis quelques années car il possède toutes les qualités des outils centralisés mais ajoute encore d'autre fonctionnalité voici les avantages des gestionnaires décentralisés face au centralisés.

- Possibilité de faire des commits atomique sur sa machine et les pousser ensemble en une seule fois sur le serveur maître pour résoudre un bug ou créer une fonctionnalité. Il est aussi partie de travailler hors ligne ce qui est totalement impossible en centralisé
- Un aspect communautaire renforcé :

  - Une jolie interface graphique où chacun peut voir et annotez-le commit de l'autre.
  - Chacun peut proposer des idées postées des bugs, etc...

- Plus de sécurité :

  - Si les droits d'accès sont bien fait le développeur doivent passer par la case "pull request" c'est-à-dire qu'il faut que son commit soit accepté par l'administrateur pour être mis en ligne sur le serveur maître.

Au vu des fonctionnalités proposé par les serveurs de sources décentralisé il est conseillé d'opter pour ceux-ci, en particulier pour Git.

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 vérifier que le code n'est pas remplis de copier-coller plus ou moins justifié. En effet la multiplication des copié-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 tous les autres morceaux où le code a été copié-coller. Ce qui rend donc le code très vite inmaintenable.

L'outil lui va aider les équipes à détecter les morceaux de code recopie ce 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 chaîne 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

Une analyse du code permet de générer des données d'analyse sur la mise en conformité par rapport aux standards du marché car la qualité d'une application est directement liée à la qualité du code et à la productivité.

De nombreux outils permettent de contrôler quelques aspects de cette qualité du code, principalement sur l'exécution de tests unitaires, l'analyse de la couverture du code par ces tests, la vérification du respect des règles de codage, etc. Ainsi ces outils permettent d'avoir une confiance accrue en son application ! Un contrôle fréquent de la qualité du code va donc pousser l'équipe de développement à adopter et à respecter certains standards de développement. Un code qui respecte ces standards est un code plus sûr car cela permet de trouver immédiatement les erreurs.

Lorsque l'on sait que le coût de la correction d'une erreur augmente considérablement avec le temps, un outil de surveillance permet la détection précoce de ces éventuels problèmes et l'on comprend très vite l'importance de la détection rapide des erreurs ...

3.6 Logiciel de suivi de problemes

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

Avant la création de ce genre d'outils la gestion de projets était bien plus difficile et l'équipe était beaucoup moins réactive puisque l'information était souvent soit non mises à jour où pire complément indisponible. Avec un bug tracker efficace c'est un jeu d'enfant de rentrer des tâches, des bugs, de voir l'état d'avancement de ses collègues. Les bugs tracker moderne proposent aussi de nombreuses autres fonctionnalités comme des forums, des diagrammes de Gantt, etc.

La majorité des bugs tracker peuvent/doivent être installés sur un de ses serveurs dans ce genre de solutions on trouve Redmine (gratuit) ou Jira (payant pour les projets non open-source) qui est de très bons 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 agiles et sont surtout là pour les projets de petite envergure (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 sites 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 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

3.10 Test GUI

3.11 Conclusion

Dans cette conclusion je souhaiterais donner un ordre d'importance à tous les outils, car en effet, ils n'ont pas tous le même intérêt ni la même importance. Je vais donc cité ici les outils vraiment indispensables pour avoir une bonne base pour son intégration continue.

Dans un premier temps il faut absolument versionner sont code si celà n'est pas fait même pas la peine de penser à l'intégration continue, l'utilisation d'un gestionnaire de sources decentralisé peut être un plus. Utiliser les outils de tests unitaires, avant d'écrire toutes sortes de tests plus ou moins utile (fonctionnel, GUI, etc...) il faut absolument écrire des tests unitaire qui valideront à chaques modifications que le code est toujours aussi stable qu'avant. Enfin il vous faudra bien évidement le serveur d'ingration continue qui sera chargé de faire le liens entre les différents outils cité précédements.

Si les outils que j'ai cité ci-dessus sont les plus important cela ne signifie pas que les autres ne sont pas utile à l'intégration continue et qu'ils ne doivent pas être installer. Cela signifie qu'ils ne sont pas necessaire pour les petites équipes ou les petits projets, ou qu'ils peuvent être facilement intégré par la suite dans le workflow de son intégration continue si on veut augmenter la qualité du code produit. En effet les outils que j'ai cité comme nécéssaire le sont surtout car il sans eux il serait impossible de parlé d'intégration continue. Il peut etre aussi intéressant de gardé uniquement très peu de technologie pour éviter de dérouter les développeurs en

[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 assomants de nouvelles contraintes, puis au fur et à mesure que le le projet grossit et que les développeurs s'habitue à incorporer de noveaux outils, en bref rendre ça ludique et leurs montrer que l'intégration continue n'est pas juste quelques choses qui sanctionne, mais aussi un support d'amélioration pour les developpeurs. Qu'y a-t-il de plus gratifiant que de voir la courbe de qualité de son projet montée ?

3.12 Les méthodes qui supportent l'integration continue

3.13 Methode agile

3.14 Le developpemnt piloté par les tests (TDD)

4 Ça va bientôt arriver dans le cloud !

4.1 Intégration continue

4.2 La livraison continue