Table des matières

[I. Intégration continue 3](#_Toc426473687)

[1. Définition 3](#_Toc426473688)

[2. Pratiques de l'intégration continue 3](#_Toc426473689)

[ Maintenir un gestionnaire de sources unique 3](#_Toc426473690)

[ Automatiser la construction (Build) 3](#_Toc426473691)

[ Rendre le Build « auto-testant » pour capturer les bugs 3](#_Toc426473692)

[ Tout le monde soumet son travail (commit) au même endroit. 4](#_Toc426473693)

[ Chaque commit doit déclencher un build sur la machine d’intégration 4](#_Toc426473694)

[ Garder le Build rapide afin d’avoir un feedback le plutôt possible 4](#_Toc426473695)

[ Tester dans un environnement qui une copie de celui de production 4](#_Toc426473696)

[ Rendre facilement disponibles les exécutables les plus récents 4](#_Toc426473697)

[ Tout le monde peut voir ce qu'il se passe 5](#_Toc426473698)

[ Automatiser le déploiement 5](#_Toc426473699)

# Intégration continue

## Définition

L'intégration continue est une pratique de développement logiciel où les membres d'une équipe intègrent leur travail fréquemment.

En général, chacun intègre au moins quotidiennement - conduisant à de multiples intégrations par jour. Chaque intégration est vérifiée par un système automatisé de construction (y compris les tests) pour détecter les erreurs d'intégration le plus rapidement possible.

Beaucoup d'équipes trouvent que cette approche conduit à une réduction considérable des problèmes d'intégration et permet à une équipe de développer un logiciel de qualité plus rapidement. « Martin Fowler »

## Pratiques de l'intégration continue

### Maintenir un gestionnaire de sources unique

Cela permet au code source de ne pas être éparpillé. Il permet également de garder un audit complet sur l’ensemble des changements qui ont eu lieu au fils des années.

Il existe plusieurs gestionnaires de source ; Git, SVN, CVS …

Le logiciel doit être construit à partir du gestionnaire de sources. Celui-ci doit être comme le lieu unique qui contient le code source de référence.

### Automatiser la construction (Build)

Le build d’un logiciel peut être un processus long et fastidieux. Son automatisation permet d’éviter les erreurs humaines (oublie de certains fichiers ou certaines étapes …) et de gagner un temps considérable que peut être alloué à des tâches à plus forte valeur ajoutée.

### Rendre le Build « auto-testant » pour capturer les bugs

Traditionnellement, un build signifie compiler, lier et toutes les autres tâches requises pour obtenir un programme exécutable.

Or un programme peut compiler mais cela ne signifie en aucun cas qu’il marche et qu’il est exempt de tout bug.

C’est pour cette raison qu’il faut associer à la phase de build une batterie de tests automatisés.

Ces tests sont notamment ce qu’on appelle les tests unitaires, des framworks tel junit et xunit aident à leur réalisation.

### Tout le monde soumet son travail (commit) au même endroit.

Le fait de faire des commits au même endroit améliore la communication. Ça permet et chaque développeur de savoir ce qu’a fait son voisin.

Avant de faire un commit le développeur doit récupérer le contenue du SCM, faire un build et exécuter les tests afin de s’assurer que son travail n’introduit pas de régression.

Il est aussi souhaitable que les développeurs fassent des commits tous les jours de manière à faciliter l’intégration de leur code. Ça permet également d’avoir des feedbacks réguliers.

### Chaque commit doit déclencher un build sur la machine d’intégration

Cela permet de s’assurer que le code est toujours dans état stable. Le but est de corriger les erreurs le plutôt possible. Ça évite leur accumulation.

Le fait de corriger tout de suite, évite au développeur de se replonger dans code ancien et de switch de contexte.

### Garder le Build rapide afin d’avoir un feedback le plutôt possible

L’intérêt de l’intégration continue est d’avoir un feedback rapide. La méthode XP (eXtreme Programming) recommande de ne pas dépasser dix minutes.

L’intérêt c’est aussi d’inclure le moins de commits possible à chaque build pour faciliter la correction en cas de bug.

### Tester dans un environnement qui une copie de celui de production

L’intérêt des tests et de trouver des bugs dans un environnement maitrisé.

Si vous testez dans un environnement différent, chaque divergence entraine un risque que ce qui arrive en cours de test n'arrivera pas en production.

### Rendre facilement disponibles les exécutables les plus récents

Toute personne participant à un projet de logiciel devrait être en mesure d'obtenir les derniers exécutables et être capable de les exécuter : pour des démonstrations, des essais exploratoires, ou juste pour voir ce qui a changé cette semaine.

### Tout le monde peut voir ce qu'il se passe

L’idée c’est de rendre visible les résultats de builds via une interface graphique out tout autre système.

Certaines équipes utilisent un système de lampes qui passent de vert en cas de succès à rouge en cas d’échec.

### Automatiser le déploiement

Pour éviter des erreurs humaines mais aussi gagner du temps. Notamment lorsqu’il s’agit de déployer les scripts de base de données ou toute autre ressource sensible.