

Bitte Platzhalter löschen
und durch eigenes Bild
ersetzen



Intégration continues

Rapport 2 - Séminaire Info 2015

Filière d'études : Informatique

Auteurs : Emanuel Knecht, David Aeschlimann

Conseiller : Dr. Bernhard Anrig

Date : 30 novembre 2015

Abstract

Table des matières

Abstract	ii
1 Introduction	2
1.1 Mission	2
1.2 Approche	2
2 Intégration Continue	3
2.1 Histoire	3
2.2 Aperçu	4
2.2.1 Architecture exemplaire	5
2.3 Concepts de l'Intégration Continue	6
2.3.1 Construction continue	6
2.3.2 Intégration continue de base de donnée	6
2.3.3 Test continue	6
2.3.4 Inspection continue	6
2.3.5 Information en retour continue	6
2.3.6 Déploiement continue	6
2.4 Motivation et bénéfices	7
2.4.1 Éviter des risques	7
2.5 Meilleures pratiques	8
3 Évaluation	9
3.1 Outils nécessaire pour CI	9
3.2 Logiciels de construction	9
3.2.1 Ant	9
3.2.2 Maven	9
3.3 Serveur de l'intégration continue	9
3.3.1 Jenkins	9
3.3.2 TeamCity	9
3.3.3 Bamboo	9
3.3.4 Hudson	9
3.4 Autre outils	9
4 Conclusion	10
4.1 Bilan	10
Bibliographie	11
Table des figures	11

1 Introduction

Ce document est la partie écrite du module Séminaire Informatique de l'Haute école spécialisée de Berne.

1.1 Mission

L'objectif de ce rapport est d'offrir un aperçu de l'intégration continue (Continuous Integration) et des solutions existantes aux lecteurs.

Dans une première partie la notion d'intégration continue et les concepts correspondants seront expliqués. De plus il faut absolument mentionner les meilleures pratiques de l'IC et les bénéfices qu'on reçoit si on décide d'implémenter les concepts et respecter ces pratiques.

Dans une deuxième partie du rapport on vous donnera une vue d'ensemble de tous les outils disponibles pour pratiquer l'IC. À cause du nombre immense de différents outils, il ne nous sera pas possible de considérer tous les composants et fournisseurs existants. Le but est de démontrer les avantages et désavantages de quelques outils sélectionnés, entre autres les outils les plus répandus.

1.2 Approche

Pour commencer, la connaissance de la matière devait être acquise et solidifiée. Dans notre parcours professionnel on avait déjà rencontré des systèmes de l'intégration continue, mais seulement comme utilisateurs et jamais comme administrateurs. Après avoir défini la structure de notre rapport on a partagé les travaux et continué à travailler individuellement. Après avoir fini la partie écrite on a corrigé le travail ensemble.

Pour être capable de démontrer différents serveurs de CI et mieux donner une évaluation, on a décidé de configurer et installer trois serveurs en nuage. De plus on a créé un projet de test en Java et C# pour illustrer un processus d'IC complet.

2 Intégration Continue

2.1 Histoire

La notion *Intégration Continue* était mentionnée dans un livre de Grady Booch en 1994 pour la première fois¹. Il parlait d'une intégration continue par des publications internes et chaque publication apporte l'application plus proche à la version finale.

La prochaine fois que l'intégration continue était sous les feux de l'actualité était avec la publication des concepts de *Extreme Programming* en forme d'un livre en 1999. Là incluse est l'idée d'avoir une machine dédiée à l'intégration du code et les pairs de développeurs réunissant, intégrant et testant le code source après chaque changement.²

Une autre personne qui a gravé la notion *Intégration Continue* est Martin Fowler. Il a publié un article sur le sujet en 2000 et révisé celui-ci six ans plus tard.³ Dans cet article il essaierait de donner une définition de l'IC et des meilleures pratiques. Martin Fowler travaillait chez ThoughtWorks l'entreprise responsable pour la publication du premier serveur d'*Intégration Continue* "Cruise Control". Il est souvent cité comme personne-clé si on parle de l'IC.

Le premier livre publié sur la matière était "Continuous Integration"⁴ en 2007. Naturellement il y a beaucoup d'autres livres traitant des technologies ou outils concrets. Aujourd'hui le plus grand nombre d'entreprises implémentent quelques ou tous les aspects de l'IC.

1. Booch (1993)
2. Roberts (2015)
3. Fowler (2006)
4. Duvall (2007)

2.2 Aperçu

Pour pouvoir comprendre le concept de base de l'Intégration Continue il est nécessaire de connaître le processus de développement logiciel ordinaire. L'Intégration Continue n'exige pas de méthode de gestion de projets spécifique mais est souvent utilisé avec des approches agiles, car elle les complètent parfaitement. Voici un diagram d'un processus pareil.⁵

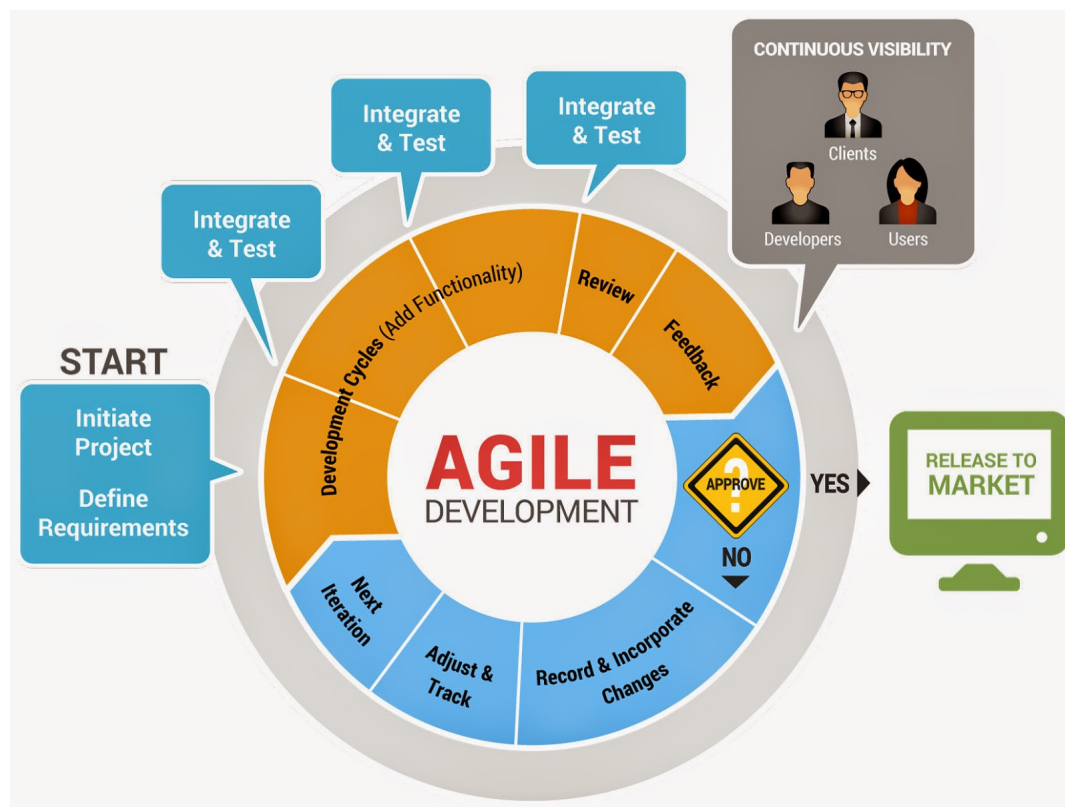


Figure 2.1 – Processus de développement logiciel

L'idée principale de L'IC est d'automatiser des devoirs ou d'offrir de l'aide pendant les étapes de développement. Pendant presque tous les projets de développement on travaille en équipe. Tous les développeurs font des changements et ajoute de la fonctionnalité chaque jour. C'est pourquoi c'est nécessaire de réunir ces changements régulièrement et de vérifier si tous les composants marche et coopère comme voulu.

Ce processus de réunification s'appelle l'intégration. Si l'intégration est faite continuellement on parle de l'**Intégration Continue**. Mais une Intégration Continue à la lettre, chaque minute ou même en temps réel, n'est pas faisable ou aidant. C'est pour ça qu'une intégration exécuter au moins une fois par jour est suffisante. De plus cette intégration doit être facile et automatisée, comme pousser un bouton. Tout le reste est contrôlé par le système IC. De plus la définition et l'étendue de l'IC est ouverte et pas strictement limitée. Mais il y a quelques éléments qui apparaissent dans tous les systèmes d'IC.

5. Source <http://www.techtipsnapps.com/2015/04/most-successful-software-development.html>

2.2.1 Architecture exemplaire

Voici une architecture normale en travaillant avec un système d'Intégration Continue.

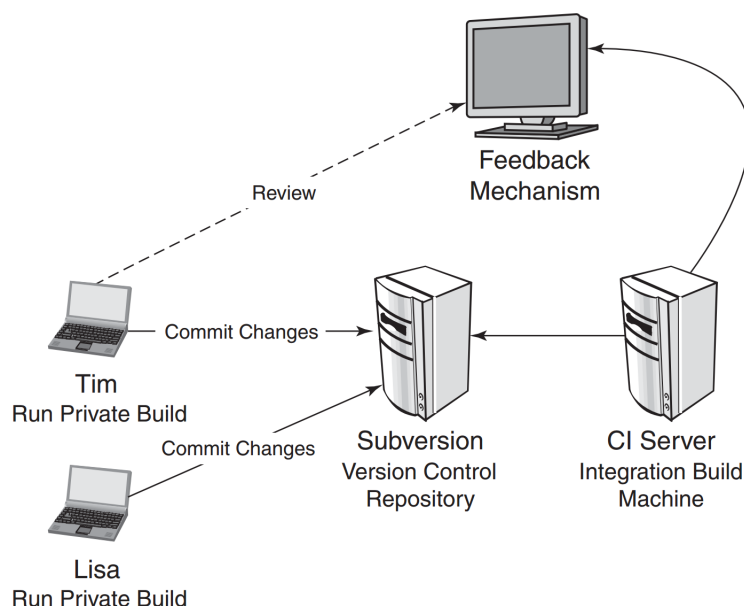


Figure 2.2 – Architecture exemplaire

Developpement locale

Tous les developpeurs travaille sur ses machines privées ou des machines de l'entreprise. Avant d'ajouter de la fonctionnalité la version la plus actuelle est téléchargé du dépôt centrale. Après changer le code il est necessaire de faire un build et d'executer les testes unitaires. Si tout fonctionne les changements sont commit sur le dépôt centrale.

Dépôt centrale

Le dépôt centrale est normalement un système de gestion de versions comme Subversion ou Git. Il est installé sur un serveur dédié. Tous les developpeur recoivent de l'accès pour commettre des changements là dessus. Ce faisant il est possible d'identifier qui a changé quoi en cas que quelque chose ne marche plus.

Serveur de l'Intégration Continue

Le serveur de l'Intégration Continue est la partie centrale du système. Il est aussi installé sur un serveur dédié, quelque foi sur la même machine que le dépôt centrale. Le serveur IC controle reulièremment s'il y a une nouvelle version dans le dépôt centrale. Si c'est le cas le serveur prends le code et démarre le processus d'intégration. Les étapes du processus doivent normalement être configurer une fois par projet. Quelques exemples pour des étapes possibles :

- Construction du code
- Testes unitaires, testes d'intégration ou testes fonctionnelle
- Inspections
- Création de sous-système ou deploiement

Si on inclue le deploiement dans le processus d'integration, il ne s'agit plus seulement de l'Intégration Continue, mais aussi du **Déploiement et de la Livraison Continue** (Continous Deployment, Continuous Delivery).

Information en retour

Après finir ce processus d'intégration les résultats des étapes est affiché sur un interface d'utilisateur centrale, souvent une page web. Pour toutes les étapes il est possible de configurer si l'échec de l'étappe amene l'échec du processus complet. Si il y a des erreurs les developpeurs et les personnes responsable peuvent être informer par des emails. Quelques entreprises installe des écrans dans les bureaux qui affichent les dernières résultats en temps réel.

Ces concepts seront discuter en détail pendant le prochain chapitre.

2.3 Concepts de l'Intégration Continue

2.3.1 Construction continue

2.3.2 Intégration continue de base de donnée

2.3.3 Test continue

2.3.4 Inspection continue

2.3.5 Information en retour continue

2.3.6 Déploiement continue

-> Continuous Delivery concept

2.4 Motivation et bénéfices

La raison principale pour utiliser l'IC est de garantir le succès et le déroulement d'un projet de développement de logiciel sans accroc. Dans tous les projets il y aura des problèmes et dans tous les logiciels il y aura des bogues. Mais l'IC aide à minimiser l'impact négatif que ces erreurs ont.

De plus l'IC fait possible d'automatiser des processus ennuyeux, répétitif et sensible aux défauts. Par ça on peut économiser du temps et de la monnaie et les développeurs se peuvent concentrer sur ce qu'ils aiment faire.

2.4.1 Éviter des risques

En dessous vous trouvez quelques risques que l'IC aide à éviter, mais seulement si la méthodologie est appliquée correctement (Meilleures pratiques).⁶

Logiciel pas déployable

Si on fait l'intégration d'un système seulement à la fin du projet, la probabilité de ne pas être capable de déployer et dérouler le logiciel pour le client est très haute. Des énoncés comme "Mais ça marche sur ma machine" sont très connus. Des raisons pour cela peuvent être des configurations manquantes ou différentes sur la machine cible, ou même des dépendances qui n'ont pas été incluses pour le déploiement. Naturellement si le code source ne compile pas, le logiciel ne peut pas être déroulé.

En commençant, construire et déployer le logiciel souvent ce risque peut être diminué. En faisant ça on a la certitude d'avoir au moins un logiciel qui marche partiellement.

Découverte tardive des erreurs

Par l'exécution des tests automatiques pendant le processus de construction des erreurs dans le code source peuvent être découvertes tôt. De plus il est aussi possible de déterminer la couverture du code par les tests. Sûrement la qualité des tests doit être bonne.

Manque de visibilité du projet

L'opération d'un serveur d'IC crée la clarté de l'état actuel de l'application et aussi de sa qualité. Si il y a un problème avec les changements derniers toutes les personnes responsables seront contactées. Si une nouvelle version a été déployée pour le testing, les personnes testant le logiciel seront aussi automatiquement informées.

Il existe même des outils qui font la visualisation du projet possible, par générant des diagrammes UML du code courant. Ça aide à donner un aperçu pour des développeurs nouveaux et garantit une documentation toujours actuelle du projet.

Logiciel de basse qualité

Le code source qui ne suit pas les règles de programmation, le code source qui suit une architecture différente ou le code redondant pourront devenir des erreurs dans le futur. Par exécution des tests et des inspections régulièrement ces dérogations peuvent être trouvées avant de devenir un vrai problème.

6. (Duvall, 2007, p39)

2.5 Meilleures pratiques

En dessous vous trouvez quelques pratiques qui aident à optimiser l'efficacité d'un système d'IC.⁷

1. **Étendue de l'implementation (Scope of implementation)**
Avant de commencer l'implementation d'un système de l'IC c'est absolument nécessaire de savoir de quelles composants on a besoin. Pas tous les projets necessite les mêmes mesures, ça dépends fortement de la taille, de la complexité du projet et du nombre de personnes impliqué. De plus il est conseillé de ne pas configurer tous les composant en même temps, mais de faire ça par étapes (p.ex. build, testing, review, deploy).
2. **Commettre le code souvent (Commit code frequently)**
Il est conseillé de commettre le code source au moins une fois par jour. Essaie de fragmenter le travail dans des morceaux petits et de commettre après chaque partie.
3. **Ne jamais commettre du code non-compilable (Dont commit broken code)**
4. **Éviter le code non-fonctionnant (Avoid getting broken code)**
5. **Faire la construction localement (Run private builds)**
6. **Découpler le processus de construction de l'IDE (Decouple build process from IDE)**
L'IDE peut faire des pas dans le processus de construction qui ne sont pas transparent pour le développeur ou les développeur utilisent des différents IDE. C'est pour ça que la construction doit être possible et faite à l'extérieur d'un IDE.
7. **Reparer des constructions non-fonctionnant immédiatement (Fix broken builds immediately)**
Si quelque chose ne marche pas la réparation doit avoir la première priorité.
8. **Écrire des tests automatisés (Write automated developer tests)**
9. **Tous les tests doivent réussir (All tests and inspections must pass)**
Si on ignore des tests qui ne réussissent pas on diminue la visibilité du projet.
10. **Des constructions vite (Keep builds fast)**

7. CI and You (Duvall, 2007, p 47)

3 Évaluation

3.1 Outils nécessaire pour CI

3.2 Logiciels de construction

3.2.1 Ant

3.2.2 Maven

3.3 Serveur de l'intégration continue

3.3.1 Jenkins

3.3.2 TeamCity

3.3.3 Bamboo

3.3.4 Hudson

TFS, Cruise Control, Travis CI, Go

3.4 Autre outils

Test coverage, coding conventions, etc

4 Conclusion

4.1 Bilan

Bibliographie

- G. Booch. *Object-Oriented Analysis and Design with Applications*, volume 1. Addison-Wesley, 1993. ISBN 978-0805353402.
- P. M. Duvall. *Continuous Integration : Improving Software Quality and Reducing Risk*, volume 1. Addison-Wesley Professional, 2007. ISBN 978-0321336385.
- M. Fowler. Continuous integration, 2006. URL <http://www.martinfowler.com/articles/continuousIntegration.html>.
- M. Roberts. 15 years of ci, 2015. URL <http://bit.ly/18EMkmW>.
- various. Continuous integration, 2015. URL https://en.wikipedia.org/wiki/Continuous_integration.

Table des figures

2.1	Processus de developpement logiciel	4
2.2	Architecture exemplaire	5