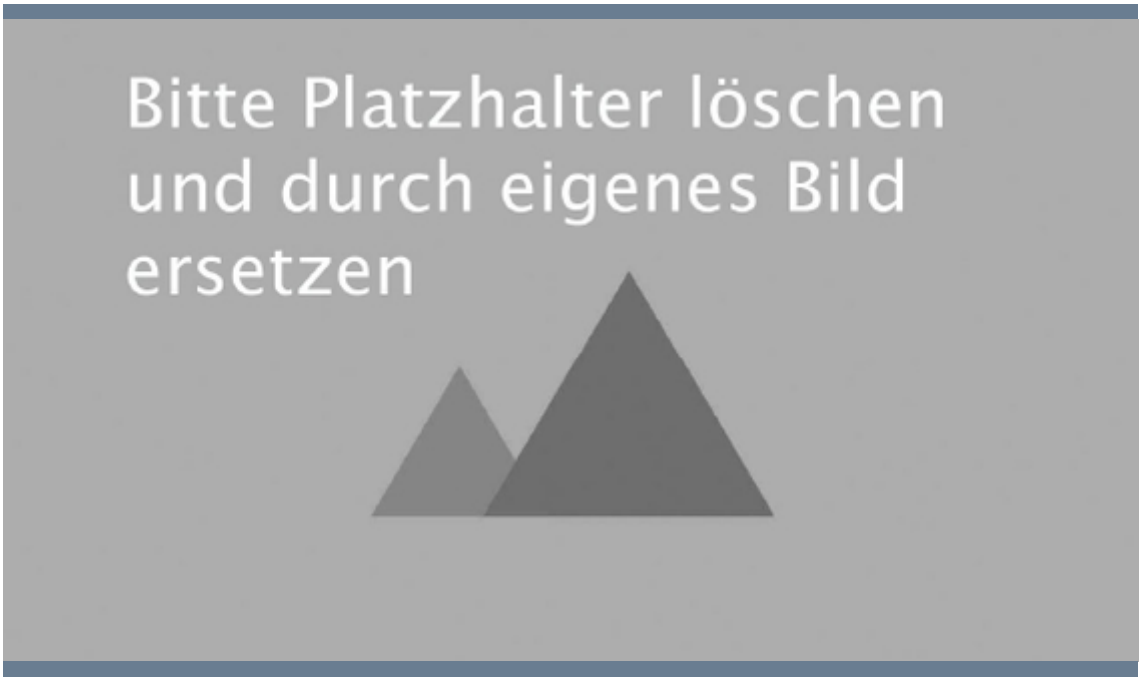

Intégration continues

Rapport 2 - Séminaire Info 2015

Filière d'études :Informatique
Auteurs : Emanuel Knecht, David Aeschlimann
Conseiller : Dr. Bernhard Anrig
Date : 23 novembre 2015

Abstract

Table des matières

1 Introduction

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

1.1 Mission

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

Dans une première partie la notion Intégration Continue et les concept correspondants seront expliquer. De plus il faut absolument mentionner les meuilleures pratiques de l'IC et les benefices qu'on recoit si on decide d'implémenter les concepts et réspecte ces pratiques.

Dans une deuxième partie du rapport on vous donneras une vue d'ensemble de touts les outils disponible pour pratiquer l'IC. à cause du nombre immense de different outils, il ne nous sera pas possible de considérer tous les composant et fournisseurs existant. Le but est de démontrer les avantages et désavantages de quelques outils sélectionné, entre autres les outil les plus répandu.

1.2 Approche

Pour commencer, la connaissance de la matière devait être acquisée et solidifiée. Dans notre parcours professionnel on avait déjà encontré des système de l'Intégration Continue, mais seulement comme utilisateurs et jamais comme administrateur. Après avoir definie la structure de notre rapport on a partagé les travaux et continué à travailler individuellement. Après avoir finit la partie écrite on a corrigé le travail ensemble.

Pour être capable de démontrer des differents serveurs de CI et mieux donner une évaluation, on a decidé de configurer et installer trois serveur en nuage. De plus on a créeé on projet de teste en java et c# pour illustrer un processus d'IC complet.

2 Intégration Continue

2.1 Aperçu

Le processus de developpement d'un logiciel. Ce que c'est CI, la definition ? Les concepts core, quels composant faut-il qu'on parle de CI ?

2.2 Histoire

La notion *Intégration Continue* etait mentionné dans un livre de Grady Booch en 1994 pour la première foi [1]. Il parlait d'une intégration continue par des publications interne et chaque publication apporte l'application plus proche à la version finale.

La prochaine foi 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 inclue est l'idée d'avoir une machine dédié à l'integration du code et les pairs de developpeurs reunissant, integrant 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 revisé celui-ci six ans plus tard. [3] Dans cet article il essayerait de donner une definition de l'IC et des meuilleures pratiques. Martin Fowler travaillait chez ThoughtWorks l'entreprise responsable pour la publication du première serveur d'*Intégration Continue* "Cruise Control".

2.3 Concepts

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 Déploiment continue

-> Continuous Delivery concept

2.3.6 Information en retour continue

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

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 auras des problèmes et dans tous les logiciels il y auras des bogues. Mais l'IC aide à minimiser l'impact negatif que cettes 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 développeur se peuvent concentrer sour ce qu'ils aiment faire. L'IC compléte parfaitement les methodes de gestion de projets agile.

2.4.1 Éviter des risques

En dessous vous trouvez quelques risques que l'IC aide à éviter, mais seulement si elle est appliquée correctement (Meuilleures pratiques). [4]

Logiciel pas déployable

Si on fait l'integration du système seulement à la fin du projét, la probabilité de ne pas être capable à déployer et dérouler le logiciel pour le client est très haute. Des énonces comme "Mais ça marche sur ma machine" sont très connues. Des raisons pour cela peuvent être des configurations manquantes ou differentes sur la machine cible, ou même des dépendances qui n'ont pas été inclus pour le deploiement. Naturellement si le code source ne compile pas, le logiciel ne peut non plus être déroulé.

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

Découverte tarde des erreurs

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

Manque de visibilité du projet

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

Il existe mêmes des outils qui font la visualisation du projet possible, par génerant des diagrams UML du code courant. Ca aide à donner un apercu pour des developpeurs nouvels et garantis une documentation toujours actuel du projet.

Logiciel de basse qualité

Le code source qui ne suit pas les règles de programmation, le code source qui suit une architecture different ou le code redondant pourrant devenir des erreurs dans le futur. Par executant des testes et des inspections regulièrement ces dérogations peuvent être trouvé avant de devenir un vrai problème.

4. (Duvall, 2007, p39)

2.5 Meuilleures pratiques

En dessous vous trouvez quelques pratiques qui aident à optimiser l'efficacité d'un système d'IC. [5]

1. **Étendue de l'implementation (Scope of implementation)**
   Avant de commencer l'implementation d'un système de l'IC c'est absolument necessaire 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 étappes (p.ex. build, testing, review, deploy).
2. **Commettre le code souvent (Commit code frequently)**
   Il est conséillé de commettre le code source au moins une foi 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-fonctionnnant (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 developpeur ou les developpeur utilises des different IDE. C'est pour ça que la construction doit être possible et fait à dehors d'une IDE.
7. **Reparer des constructions non-fonnctionnant immédiatement (Fix broken builds immediately)**
   Si quelque chose ne marche pas la reparation doit avoir la première priorité.
8. **Écrire des testes automatisé (Write automated developer tests)**
9. **Tous les testes doivent réussir (All tests and inspections must pass)**
   Si on ignore des testes qui ne réuissent pas on diminue la visibilité du projét.
10. **Des constructions vite(Keep builds fast)**

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

3 Évaluation

3.1 Outils necessaire 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