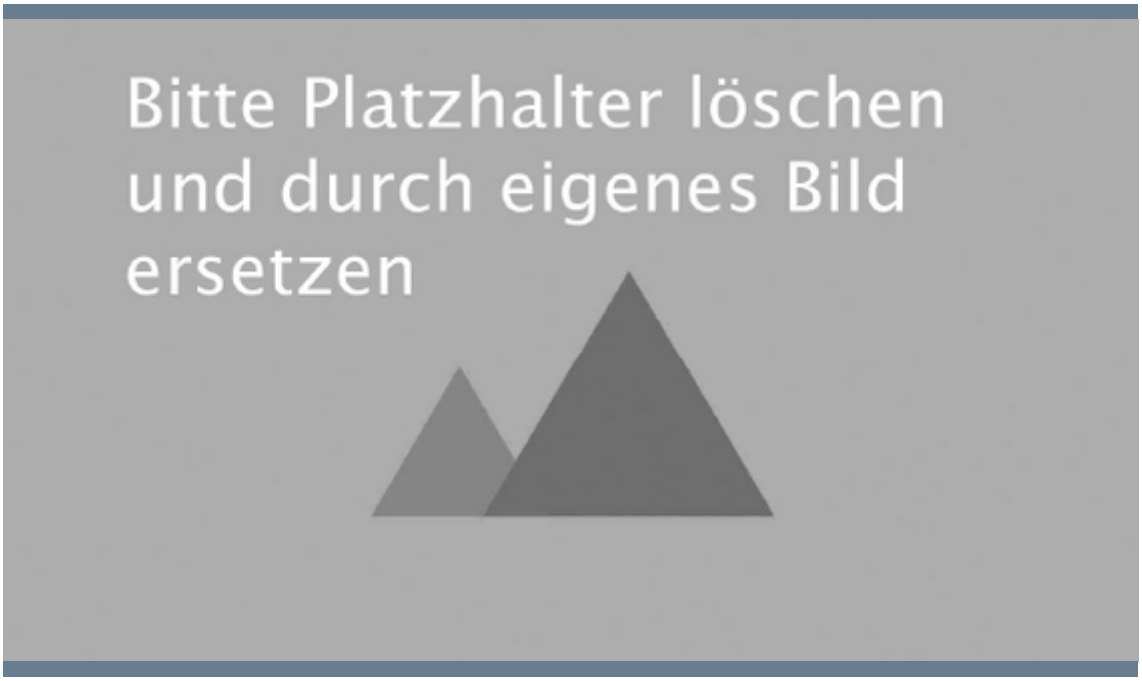

Intégration continues

Rapport 2 - Séminaire Info 2015

Filière d'études :Informatique
Auteurs : Emanuel Knecht, David Aeschlimann
Conseiller : Dr. Bernhard Anrig
Date : 7 d ecembre 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 Architécture exemplaire . . . 5
- 2.3 Concepts de l'Intégration Continue . . . 6
  - 2.3.1 Construction continue . . . 6
  - 2.3.2 Intégration continue de base de données . . . 6
  - 2.3.3 Test continue . . . 7
  - 2.3.4 Inspection continue . . . 7
  - 2.3.5 Information en retour continue . . . 7
  - 2.3.6 Déploiment continue . . . 7
- 2.4 Motivation et bénéfices . . . 8
  - 2.4.1 Éviter des risques . . . 8
- 2.5 Meuilleures pratiques . . . 9

3 Évaluation 10
- 3.1 Outils necessaire pour CI . . . 10
- 3.2 Logiciels de construction . . . 10
  - 3.2.1 Ant . . . 10
  - 3.2.2 Maven . . . 10
- 3.3 Serveur de l'intégration continue . . . 10
  - 3.3.1 Apercu de l'évaluation . . . 10
  - 3.3.2 Jenkins . . . 10
  - 3.3.3 TeamCity . . . 10
  - 3.3.4 Travis CI . . . 10
  - 3.3.5 Team Foundation Server . . . 10

4 Conclusion 11
- 4.1 Bilan . . . 11

Bibliographie 12

Table des figures 12

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 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". Il est souvent cité comme personne-clé si on parle de l'IC.

Le premier livre publié sur la matière était "Continous Integration" [4] en 2007. Naturellement il y a beaucoup d'autre livres traitant des technologie ou outils concrètes. Aujourd'hui le plus part d'entreprises implementes 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 necessaire de connaitre le processus de development logiciel ordinaire. L'Intégration Continue n'éxige pas de méthode de gestion de projets specifique mais est souvent utilisé avec des approches agiles, car elle les complètent parfaitement. Voici un diagram d'un processus pareil. [5]

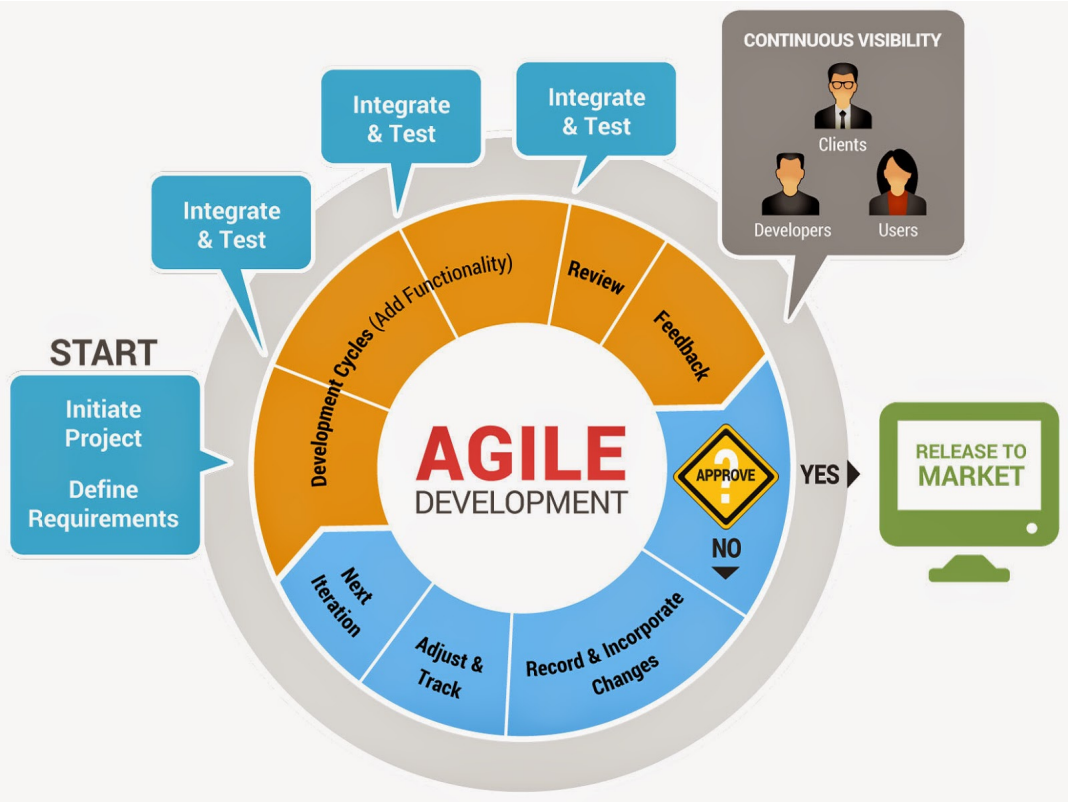

Figure 2.1 – Processus de developpement logiciel

L'idée principale de L'IC est d'automatiser des devoirs ou d'offrir de l'aide pendants les étappes de developpment. Pendant presque tous les projets de developpement on travaille en équippe. Tous les développeurs font des changements et ajoute de la fonctionnalité chaque jour. C'est pourquoi c'est necessaire de réunir ces changements regulièrement et de verifier si tous les composants marche et coopére 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 temp réel, n'est pas faisable ou aidant. C'est pour ça qu'une intégration executer au moins une foi par jour est suffisante.
De plus cette intégration doit être facile et automatisée, comme pousser un bouton. Tous le reste est controlé par le système IC. De plus la définition et l'étendue de l'IC est ouverte et pas strictement limité. Mais il y a quelques elements qui apparaissent dans tous les systèmes d'IC.

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

2.2.1 Architécture exemplaire

Voici une architécture normale en travaillant avec un système d'Integration Continue.

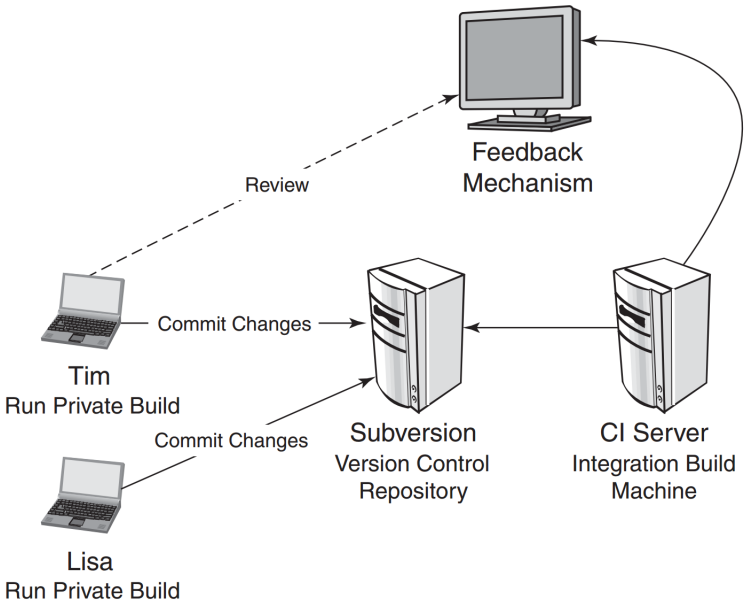

Figure 2.2 – Architécture exemplaire

Developpement locale
Tous les developpeurs travaille sur ses machines privées ou des machines de l'entreprise. Avant d'ajouter de la fonctionalité la version la plus actuelle est téléchargé du dépot 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 dépot centrale.

Dépot centrale
Le dépot 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 dédie, quelque foi sur la même machine que le dépot centrale. Le serveur IC controle reulièrement s'il y a une nouvelle version dans le dépot centrale. Si c'est le cas le serveur prends le code et démarre le processus d'intégration. Les étappes du processus doivent normalement être configurer une fois par projet. Quelques exemples pour des étappes possibles :

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

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

Information en retour
Après finir ce processus d'intégration les résultats des étappes est affiché sur un interface d'utilisateur centrale, souvent une page web. Pour toutes les étappes 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 réele.

Ces concepts seront discuter en détail pendant le prochain chapitre.

2.3 Concepts de l'Intégration Continue

2.3.1 Construction continue

Ema

2.3.2 Intégration continue de base de données

Le plus part de logiciel utilisent une méthode pour persister des données, beaucoup de fois des bases de données. Le code source pour générer cette base de données doit être traiter comme tous le reste du code d'un projet. C'est necessaire qu'il soit aussi commit sur le dépot centrale et qu'on teste et fait des inspections là dessus.

Automatisation de l'intégration

Si le code de la base de données est aussi mis sur le dépot centrale, le processus de l'intégration de base de données peut être automatisé. Ce processus peut devenir assez complexe avec different environnements. Les serveurs, les noms d'utilisateur, les mots de passe ou bien les données de test ou de système peuvent differer pour chaque environnement.

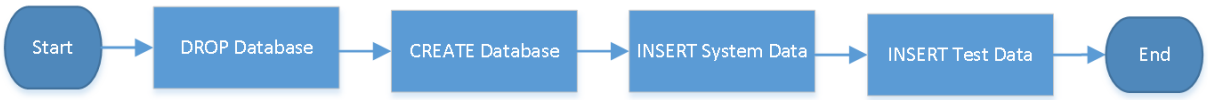

Figure 2.3 – Processus de l'intégration de base de données

C'est pour ca qu'une automatisation de ce processus est indispensable. Une automation est réalisé en insérant une section dans le script de construction uniquement pour les operations de l'intégration de base de données. L'interval et l'étendue de l'éxécution de ce processus sont pas les mêmes pour tous les projets. Pour quelques projets ca pourrait être trop lourds de recrée la base de données à chaque changement sur le dépot centrale. Avec l'outil de construction Maven et le plug-in "sql-maven-plugin" il est possible de definir quelle base de données est visé et quelles commandes sont exécuté. Voici une partie d'un tel script. [6]

```
<execution>
        <id>create-schema</id>
        <phase>process-test-resources</phase>
        <goals>
                <goal>execute</goal>
        </goals>
        <configuration>
                <autocommit>true</autocommit>
                <srcFiles>
                        <srcFile>src/main/sql/your-schema.sql</srcFile>
                </srcFiles>
        </configuration>
</execution>
```

Instance de base de donnée locale

Pour que cette approche fonctionne tous les développeurs doivent avoir l'autorisation de changer la base de données. Pour éviter des conflits pendant le développement tous les développeurs ont besoin d'une instance de cette base de données sur ses machines locales. Si on travaille avec une instance centrale à chaque changement il y a le danger de casser le code que quelqu'un d'autre est en train de développer.

6. Pour l'éxample complète SQLMavenPlugin (2015)

2.3.3 Test continue

Ema

2.3.4 Inspection continue

La difference entre les testes et les inspections est que les inspections analyse la forme et la structure du code source et pas la fonctionnalité. Ces inspections sont introduit dans le processus de construction par differents plug-in. Les inspections ne remplacent pas les code review manuelles, mais dans ces reviews il y aura moins de défauts banale à traiter. Les objectifs des inspections sont là dessous.

1. *Réduire la complexité du code source*
   La complexité du code source peut être mesurée par la métrique "Cyclomatic Complexity Number (CCN)", qui compte le nombre de chemins distincts dans une méthode. Comme ça les endroits qui nécessite un changement peuvent être identifié (Maven Plugin JavaNCSS, PMD). $http://www.mojohaus.org/javancss-maven-plugin/$
2. *Determiner la dépendance*
   Les métriques de couplage (Afferent/Efferent Coupling) et l'instabilité d'un package de logiciel peuvent être des indications à comme important on package est. Les packages qui sont utilisé très souvent il vaut mieux les tester très exactement (JDepend).
3. *Imposer les standards de l'entreprise*
   Dans chaque entreprise il y a des régles comment il faut écrire le code. Des examples très fréquemment sont que les variables n'ose pas avoir des noms trop courts et non-descriptive ou que les déclarations conditionel doivent toujours être écrit avec des parenthèses (PMD).
4. *Reduire le code copié*
   Le code source copié doit être avouer. Il y a des outils qui identifie des sections de code identique (PMD-CMD).
5. *Determiner la couverture de code*
   La couverture de code par les testes est une métrique qui aide à determiner quelles partie du code ont été negliger pendant écrire les testes (Cobertura).

Picture of interface with results or smth

2.3.5 Information en retour continue

Ema

2.3.6 Déploiment continue

Ema -> 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 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.

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 (Meuilleures pratiques). [7]

Logiciel pas déployable

Si on fait l'integration d'un 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.

7. (Duvall, 2007, p39)

2.5 Meuilleures pratiques

En dessous vous trouvez quelques pratiques qui aident à optimiser l'efficacité d'un système d'IC. [8]

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)**

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

3 Évaluation

3.1 Outils necessaire pour CI

Logiciel de construction Source Control CI Server Plugins for CI Server

3.2 Logiciels de construction

3.2.1 Ant

3.2.2 Maven

3.3 Serveur de l'intégration continue

Les critères de l'évaluation

1. Nombre d'utilisateurs
2. Nombre de plug-ins
3. Nombre de langages supporté
4. Complexité de l'installation
5. Integration avec d'autre outils de développement
6. Couts , modèle de licence (Open Source)

3.3.1 Apercu de l'évaluation

Product	Nombre d'utilisateurs	Nombre de plugin
Jenkins	alfred.kaufmann@bfh.ch	Auftraggeber, Projektleitung, Ergänzungen
TeamCity	in Pension	Tipps zur Struktur und Layout
Travis CI	in Pension	Tipps zur Struktur und Layout
TFS	ausgetreten	Erstellung der Vorlage

Table 3.1 – Serveurs de l'IC

3.3.2 Jenkins

3.3.3 TeamCity

3.3.4 Travis CI

3.3.5 Team Foundation Server

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`.

SQLMavenPlugin. Sql maven plugin, 2015. URL `http://www.mojohaus.org/sql-maven-plugin/examples/execute.html`.

various. Continuous integration, 2015. URL `https://en.wikipedia.org/wiki/Continuous_integration`.

Table des figures

2.1 Processus de developpement logiciel 4
2.2 Architécture exemplaire 5
2.3 Processus de l'intégration de base de données 6