



# Intégration continues

Rapport 2 - Séminaire Info 2015

Filière d'études : Informatique

Auteurs : Emanuel Knecht, David Aeschlimann

Conseiller : Dr. Bernhard Anrig

Date : 26 décembre 2015

# Abstract

# Table des matières

<b>Abstract</b>	<b>ii</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Mission . . . . .	2
1.2 Approche . . . . .	2
<b>2 Intégration Continue</b>	<b>3</b>
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 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 Déploiement continue . . . . .	7
2.4 Motivation et bénéfices . . . . .	8
2.4.1 Éviter des risques . . . . .	8
2.5 Meilleures pratiques . . . . .	9
<b>3 Évaluation</b>	<b>10</b>
3.1 Logiciels de construction . . . . .	10
3.1.1 Ant . . . . .	10
3.1.2 Maven . . . . .	10
3.2 Serveur de l'intégration continue . . . . .	10
3.2.1 Définition des critères . . . . .	10
3.2.2 Aperçu des resultats . . . . .	11
3.2.3 Jenkins . . . . .	12
3.2.4 TeamCity . . . . .	12
3.2.5 Travis CI . . . . .	13
3.2.6 Team Foundation Server . . . . .	14
<b>4 Conclusion</b>	<b>15</b>
4.1 Bilan . . . . .	15
<b>Bibliographie</b>	<b>16</b>
<b>Table des figures</b>	<b>16</b>

# 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 fois<sup>1</sup>. 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.<sup>2</sup>

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.<sup>3</sup> 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"<sup>4</sup> en 2007. Naturellement il y a beaucoup d'autres livres traitant des technologies ou outils concrets. Aujourd'hui le plus part 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.<sup>5</sup>

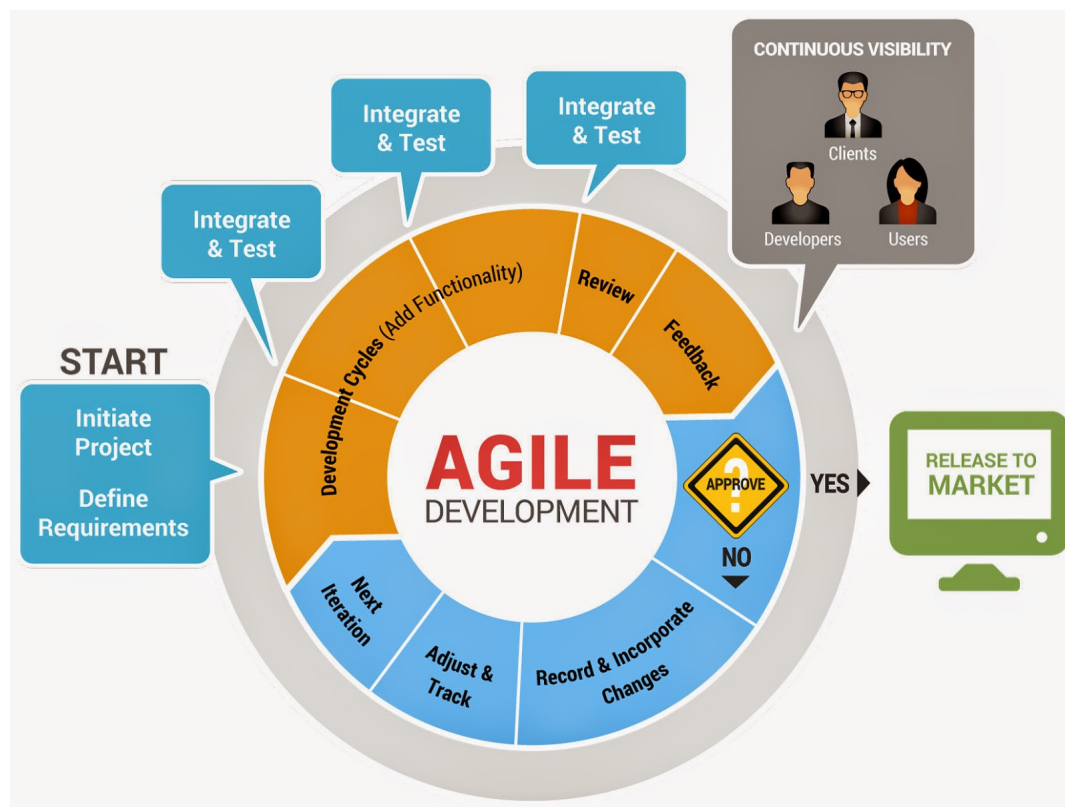


Figure 2.1 – Processus de développement logiciel

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. Tous le reste est contrôlé 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 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.

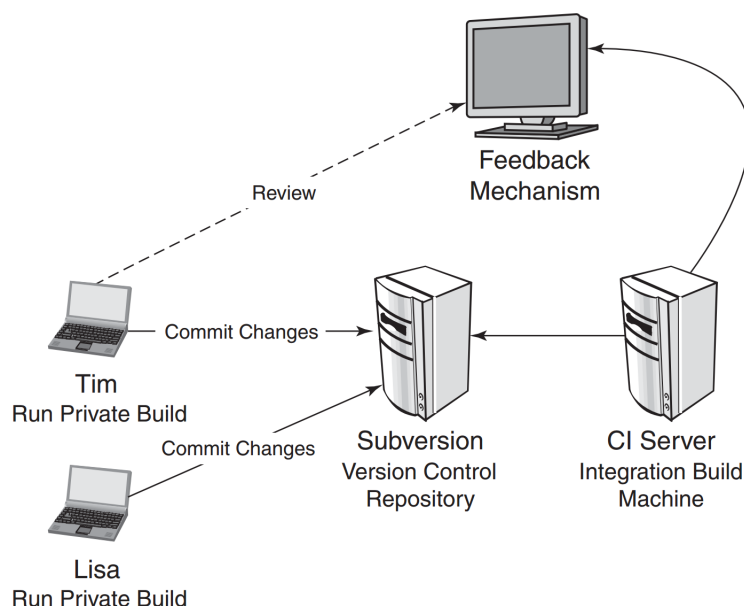


Figure 2.2 – Architecture exemplaire

### 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 reulièrément 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 fonctionelle
- 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 réelle.

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 traité comme tous le reste du code d'un projet. C'est nécessaire qu'il soit aussi commit sur le dépôt 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 dépôt centrale, le processus de l'intégration de base de données peut être automatisé. Ce processus peut devenir assez complexe avec différents environnements. Les serveurs, les noms d'utilisateur, les mots de passe ou bien les données de test ou de système peuvent différer pour chaque environnement.

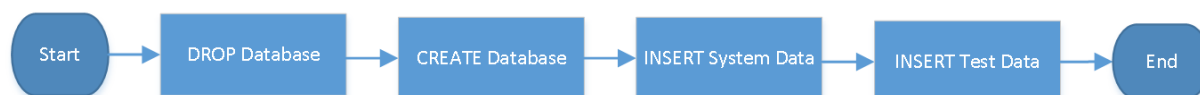


Figure 2.3 – Processus de l'intégration de base de données

C'est pour ça qu'une automatisation de ce processus est indispensable. Une automatisation est réalisée en insérant une section dans le script de construction uniquement pour les opérations de l'intégration de base de données. L'intervalle et l'étendue de l'exécution de ce processus sont pas les mêmes pour tous les projets. Pour quelques projets ça pourrait être trop lourds de recréer la base de données à chaque changement sur le dépôt centrale. Avec l'outil de construction Maven et le plug-in "sql-maven-plugin" il est possible de définir quelle base de données est visée et quelles commandes sont exécutées. Voici une partie d'un tel script.<sup>6</sup>

```
<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'exemple complète SQLMavenPlugin (2015)



### 2.3.3 Test continue

Ema

### 2.3.4 Inspection continue

La différence entre les tests et les inspections est que les inspections analysent la forme et la structure du code source et pas la fonctionnalité. Ces inspections sont introduites dans le processus de construction par différents plug-in. Les inspections ne remplacent pas les revues de code manuelles, mais dans ces revues il y aura moins de défauts banals à traiter. Les objectifs des inspections sont énumérés ci-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 nécessitent un changement peuvent être identifiés (JavaNCSS, PMD<sup>7</sup>).
2. *Déterminer la dépendance*  
Les métriques de couplage (Afferent/Efferent Coupling) et l'instabilité d'un package de logiciel peuvent être des indications à quel point un package est important. Les packages qui sont utilisés très souvent valent mieux être testés très exactement (JDepend<sup>8</sup>).
3. *Imposer les standards de l'entreprise*  
Dans chaque entreprise il y a des règles sur comment il faut écrire le code. Des exemples très fréquents sont que les variables ne doivent pas avoir des noms trop courts et non-descriptifs ou que les déclarations conditionnelles doivent toujours être écrites avec des parenthèses (PMD).
4. *Réduire le code copié*  
Le code source copié doit être évité. Il y a des outils qui identifient des sections de code identiques (PMD).
5. *Déterminer la couverture de code*  
La couverture de code par les tests est une métrique qui aide à déterminer quelles parties du code ont été négligées pendant l'écriture des tests (Cobertura<sup>9</sup>).

TODO : Picture of interface with results or smth

### 2.3.5 Information en retour continue

Ema

### 2.3.6 Déploiement continu

Ema -> Continuous Delivery concept

---

7. (PMD, 2015)

8. (JDepend, 2015)

9. (Cobertura, 2015)

## 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).<sup>10</sup>

#### 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.

---

10. (Duvall, 2007, p39)

## 2.5 Meilleures pratiques

En dessous vous trouvez quelques pratiques qui aident à optimiser l'efficacité d'un système d'IC.<sup>11</sup>

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)**

---

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

# 3 Évaluation

## 3.1 Logiciels de construction

### 3.1.1 Ant

### 3.1.2 Maven

## 3.2 Serveur de l'intégration continue

### 3.2.1 Definition des critères

En dessous vous trouvez les critères avec une bref description qui seront utilisé pour evaluer les quatres serveurs de l'intégration continue qui ont été choisis pour ce travail. <sup>1</sup>

#### 1. Caractéristiques du produit

*Les caractéristiques du produit sont l'aspect le plus important quand on choisit un serveur d'IC. On doit savoir les exigences qu'une entreprise a et de ce point de vue selectionner un logiciel.*

- Intégration avec des outils de gestion des versions  
*Est l'outil que nous utilisons supporté ? Quelles outils sont supporté ?*
- Intégration avec l'outil de construction  
*Est notre langage de programmation (compilateur) et notre outil de construction supporté ?*
- Information en retour  
*Quelles méthodes de l'information en retour existe et sont ils suffisant pour nous ?*
- Labeling  
*Est-il possible de donner des identifiants à des versions d'un logiciel ?*
- Extensibilité  
*Est-il possible d'écrire des extension propre pour le serveur si necessaire ?*

#### 2. Générale

- Fiabilité et longévité
- Environnement cible
- Infrastructure
- Coûts
- Type de logiciel

#### 3. Taille de la communauté

- Nombre d'utilisateurs
- Nombre de plug-ins

#### 4. Utilisation

- Facilité d'utilisation
- Complexité de l'installation

---

1. IBM (2006)

### 3.2.2 Aperçu des resultats

Critères	Jenkins	TeamCity	Travis CI	Team Foundation Server
<b>Caractéristique du produit</b>				
Outils de gestion des versions	Subversion/CVS(+plugins)	Subversion/CVS(+plugins)	github/Git	Git/TFVC
Outils de construction			+	
Information en retour			+	
Labeling			-	
Extensibilité	++	+	-	-
<b>Générale</b>				
Fiabilité et longévité	++	++	++	++
Environnement cible	tous	tous	Linux	Microsoft Windows
Infrastructure	On-premises	On-premises	On-premises/SaaS	On-premises/SaaS
Coûts	gratuit	Freemium*	Freemium*	Freemium
Type de logiciel	Open Source (MIT)	Propriétaire	Open Source (MIT)	Propriétaire
<b>Taille de la communauté</b>				
Nombre d'utilisateurs	many	many	medium	many
Nombre de plugins	++	+	-	-
<b>Utilisation</b>				
Facilité d'utilisation	many	many	facile	many
Complexité de l'installation	many	many	facile	many

Table 3.1 – Serveurs de l'IC

Freemium = C'est gratuit pour la version base, mais ça coûte pour des éditions plus grande (entreprise).

\* free for open source projects<sup>2</sup>

2. (Jenkins, 2015) (TeamCity, 2015) (TFS, 2015)

### **3.2.3 Jenkins**

### **3.2.4 TeamCity**

### 3.2.5 Travis CI

**Caractéristique du produit** Travis CI est un serveur de l'intégration continue très facile à mettre en service et avec une intégration excellente avec github.com. Il supporte beaucoup de différent langage et outil de construction. La liste complète peuvent être trouver dans la documentation<sup>3</sup>. Mais il seulement supporte git comme outil de gestion des versions.

De plus il offre multiple voies d'information en retour. Le plus facile est par email, mais il y a aussi la possibilité d'envoyer des messages par IRC, Slack, HipChat etc.<sup>4</sup>. Chaque construction recoit un identificateur numerique, mais il n'est pas possible de le changer. Il y a une API pour accéder à Travis (SaaS), mais l'extensibilité semble limité.



Figure 3.1 – Travis CI Logo

**Générale** Il existe trois version de Travis CI.

- Travis CI for Open Source
- Travis Pro
- Travis Enterprise

Les deux première versions sont accessible comme SaaS. Une version est pour des projets Open Source qui est gratuite et l'autre est pour des projets avec un dépôt de github privée qui coute. La troisième version est pour des entreprises qui veulent mettre l'infrastructure comme des serveurs à disposition eux-mêmes.

**Taille de la communauté** Travis CI affirme sur la page web qu'il y a 246506 projets Open Source qui sont testé et intégré sur leur platform. Sur la nombre d'utilisateurs des deux versions commercial il n'y a pas d'information.

**Utilisation** L'utilisation de Travis CI est très pratiques est facile. Si on a déjà un dépôt sur github, il faut seulement trois pas pour lancer l'intégration continue avec Travis.

1. Login avec le compte de github sur travis
2. Choisir le dépôt
3. Écrire un fichier .travis.yml pour définir la configuration

Après ça chaque fois qu'il y a un changement sur le dépôt les testes et la construction seras executé automatique-ment. Pour construire notre projet de test en java (maven), d'executer les testes avec trois différent version de java et envoyer un email à une adresse si quelque chose ne marche pas, le fichier en bas suffisait. De plus vous trouver un aperçu de l'interface d'administration de travis.

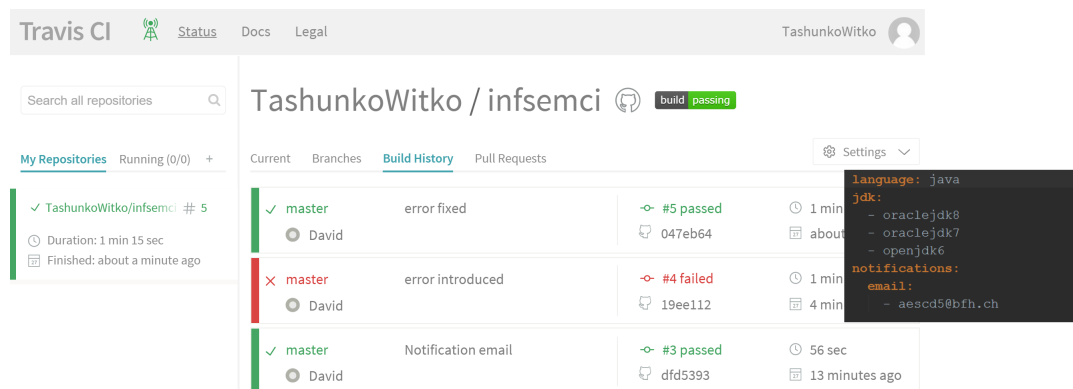


Figure 3.2 – Travis interface d'administration et yml fichier

3. (?)
4. traviscinotification

### **3.2.6 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.
- Cobertura. Cobertura - code coverage plugin, 2015. URL <http://cobertura.github.io/cobertura/>.
- 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>.
- IBM. Selection of ci server, 2006. URL <http://www.ibm.com/developerworks/library/j-ap09056/>.
- JDepend. Jdepend - design quality metrics, 2015. URL <http://www.clarkware.com/software/JDepend.html>.
- Jenkins. Plugins jenkins, 2015. URL <https://wiki.jenkins-ci.org/display/JENKINS/Plugins>.
- PMD. Pmd - code inspection plugin, 2015. URL <https://pmd.github.io/>.
- 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>.
- TeamCity. Teamcity environment, 2015. URL <https://confluence.jetbrains.com/display/TCD9/Supported+Platforms+and+Environments>.
- TFS. Tfs version control, 2015. URL <https://msdn.microsoft.com/en-us/Library/vs/alm/code/overview>.
- various. Continuous integration, 2015. URL [https://en.wikipedia.org/wiki/Continuous\\_integration](https://en.wikipedia.org/wiki/Continuous_integration).

# Table des figures

2.1	Processus de developpement logiciel . . . . .	4
2.2	Architecture exemplaire . . . . .	5
2.3	Processus de l'intégration de base de données . . . . .	6
3.1	Travis CI Logo . . . . .	13
3.2	Travis interface d'administration et yml fichier . . . . .	13