Intégration continues

Rapport 2 - Séminaire Info 2015

Filière d'études :Informatique
Auteurs : Emanuel Knecht, David Aeschlimann
Conseiller : Dr. Bernhard Anrig
Date : 9 janvier 2016

Abstract

Développer des applications stables sans effets secondaires imprévues est assez difficile. Si on a la chance de travailler dans une équipe il y a beaucoup des problèmes à résoudre, pas seulement affectant la logique du code. Supposant Jean a écrit une classe qui dépends sur une méthode définit dans un autre fichier et quelques jours plus tard Pierre change la signature ou la fonctionnalité de cette méthode avant que Jean a mis a jour ses changements. Le résultat : Une grande misère. Pour éviter des problèmes comme ca il faut un architecte expérience et un système d intégration continue.

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 aperçu 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 expliqué. De plus les meilleures pratiques seront présenté et les bénéfices qu'on reçoit si on implémente les concepts et respecte les meilleures pratiques.

Dans la 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 différent 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 acquis et solidifiée. Dans notre parcours professionnel on avait déjà rencontré des systèmes de l'intégration continue, mais seulement comme utilisateurs et jamais comme administrateur. Après avoir définie la structure de notre rapport on a partagé les travaux et continué à travailler individuellement.

Pour être capable de donner une évaluation des serveurs d'IC choisit et mieux les connaitre, on a décidé d'installer, configurer et tester chacun. Pour ces testes on a créé des projets très simples en C++ et Java avec des testes unitaires. Ces projets ont était intégré par les serveurs IC.

Après avoir finit la partie écrite, on a relu et corrigé le rapport ensemble.

2 Intégration Continue

2.1 Histoire

La notion *Intégration Continue* était 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'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 essayerait de donner une définition de l'IC et des meilleures 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 implémentes 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 connaitre 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 diagramme d'un processus pareil. [5]

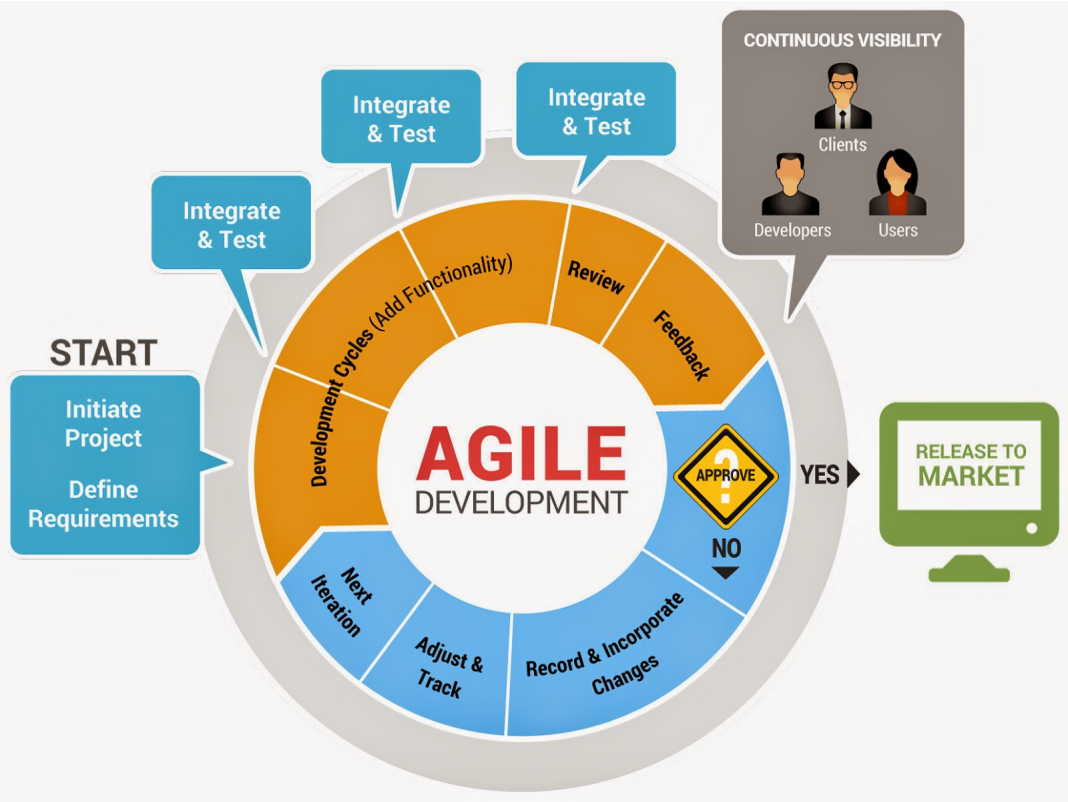

Figure 2.1 – Processus de développement logiciel

L'idée principale de L'IC est d'automatiser des devoirs des développeurs ou d'offrir des aides pendants les étapes du processus 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 pour çela qu'il est nécessaire de réunir ces changements régulièrement et de vérifier si tous les composants marche et coopère comme voulu (testes).

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 foi par jour est normalement considéré suffisante, naturellement le plus souvent le mieux.
De plus cette intégration doit être facile et automatisée, comme pousser un bouton. Après lancer l'intégration tous le reste doit être contrôlé par le système IC. 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.

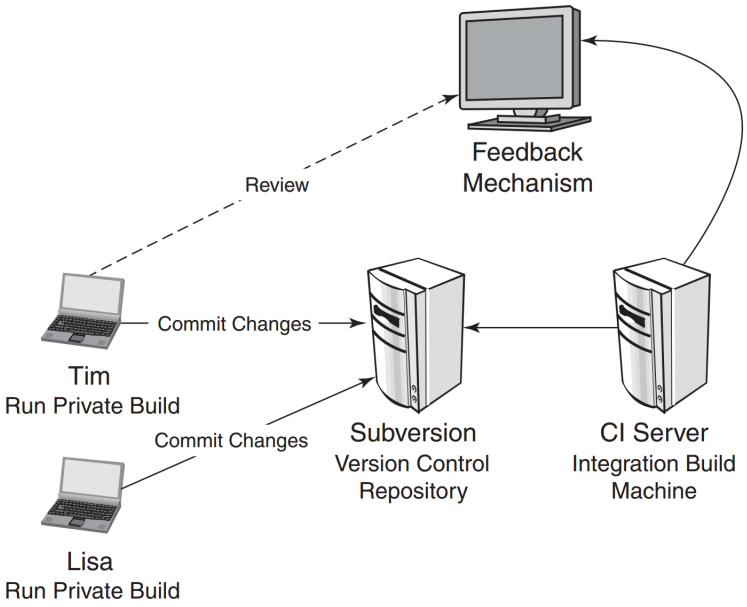

Figure 2.2 – Architecture exemplaire

Développement locale
Tous les développeurs 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 nécessaire de faire une construction et d'exécuter les testes unitaires localement. 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 développeur reçoivent 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 dépôt centrale. Le serveur IC contrôle régulièrement 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 déploiement

Si on inclue le déploiement dans le processus d'intégration, 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'étape amène l'échec du processus complet. Si il y a des erreurs les développeurs 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.

2.3 Concepts de l'intégration Continue

2.3.1 Construction continue

C/C++

Le premier logiciel de construction était make, qui existes depuis 1976 (Stuart Feldman, Bell Labs). Sur les plateformes basé sur Unix make est encore utilisé pour construire des exécutables. Pour construire une application avec make if faut fournir un fichier makefile qui contiens les instructions de compilation. Supposant on a un logiciel avec les fichiers :

Listing 2.1 – functions.h

```
int factorial(int i);
int squared(int i);
```

Listing 2.2 – factorial.c

```
#include "functions.h"
int factorial(int i)
{
        return i==1 ? 1 : i*factorial(i-1);
}
```

Listing 2.3 – squared.c

```
#include "functions.h"
int squared(int i)
{
        return i*i;
}
```

Listing 2.4 – main.cpp

```
#include <iostream>
#include "functions.h"
int main(char *arg_v, int arg_c)
{
        cout<<"Factorial 5 Squared = "<<squared(factorial(5))<<endl;
}
```

La commande pour compiler ce projet manuellement est :

$$g++ \; main.cpp \; squared.cpp \; factorial.cpp \; -o \; hello$$

Pour un projet si petit c'est assez simple mais avec plus des fichiers ça se confus très vite. Par exemple si on a beaucoup des dépendances ou une application multi-plateforme c'est très difficile de ne pas oublier quelque chose.

Listing 2.5 – Makefile sans variables

```
all: hello

hello: main.o factorial.o squared.o
    g++ main.o factorial.o squared.o -o hello

main.o: main.cpp
    g++ -c main.cpp

factorial.o: factorial.c
    g++ -c factorial.c

squared.o: squared.cpp
    g++ -c squared.cpp

clean:
    rm *o hello
```

Si on regarde le fichier Makefile correspondant on voit que ça ne simplifie pas encore notre vie. Une prochaine étape est d'introduire des variables.

Listing 2.6 – Makefile avec variables

```
CC=g++
CFLAGS=-c -Wall
SOURCES=main.cpp factorisl.c squared.c
OBJECTS=OBJECTS=$(SOURCES:=.o)
EXECUTABLE=hello

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(CC) $(OBJECTS) -o $@

.o:
    $(CC) $(CFLAGS) $< -o $@
```

Ce fichier est plus court et très adaptable, mais il n est pas très lisible. Utilisant l'outil cmake on peut générer les fichiers Makefile automatiquement (et autres fichiers de projet comme ".sln"). Dans le fichier CMakeLists.txt on définit quelles fichiers il faut compiler, quel sont les descendances...

Listing 2.7 – CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)

file(GLOB helloworld_SRC
    "*.h"
    "*.c"
    "*.cpp"
)

add_executable(hello ${helloworld_SRC})
```

Java

Ant (Another neat tool) a apparu 2000, le premier logiciel de construction pour Java. Aujourd'hui Ant et ses produits de concurrence Maven(2002) et Gradle(2012) sont inévitable si on utilise Java. Dans ce moment environ 70 % des développeurs Java utilisent Maven, 15 % Ant et 15% Gradle. Le concept est pareille comme en C++.

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 nécessaire 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 différent 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.

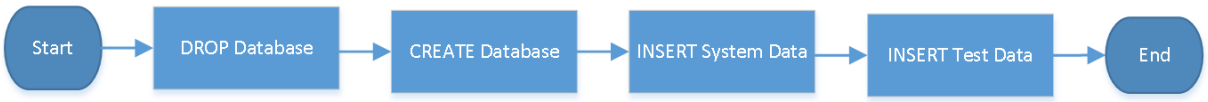

Figure 2.3 – Processus de l'intégration de base de données

C'est pour ça qu'une automatisation de ce processus est indispensable. Une automation est réalisé 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 recrée la base de données à chaque changement sur le dépôt centrale. Avec l'outil de construction Maven et le plugin "sql-maven-plugin" il est possible de définir 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'exemple complète SQLMavenPlugin (2015)

2.3.3 Test continue

Le premier principe de base pour des tests automatisés est "Fail fast" (Échouer vite). C'est pourquoi on distingue trois types différents. Comme réglé d'or on peut dire que les test unitaires prend des secondes, les tests d'intégration prend des minutes et les tests d'acceptation prend des heures.

Tests unitaires

Les tests unitaires vérifient le bon fonctionnement d' un bout de code. Pour les parties du code qui ne sont pas testable atomiquement on introduit des objets mock qui permettent des tests de fonctionnement vite et sans effets secondaires.

Tests d'intégration

Pour tester l' interaction des parties testé avec les tests unitaires. Aussi l'interaction avec le système de fichier ou une base de données est testé ici. Après les tests d'intégration les erreurs logiques doivent être éliminé.

Tests d'acceptation automatisées

Il y a différents types de test d' acceptation :

- Tests d'interface utilisateur codé
  Pour ces tests il faut enregistrer des scénario. Les outils de test permettent de cliquer des boutons, entrer de text etc.
- Tests de performance
  Ces tests montrent quelles parties du code prennent la plupart des ressources.
- Tests d'exploitabilité
  Si le logiciel utilise une base de données on peut par exemple tester la vulnérabilité contre des attaques injection SQL.
- Tests de charge
  Par exemple : On a un magasin en ligne qui doit résister 100 utilisateurs à n'importe quel moment, testé avec les scénarios "login", "recherche des articles", "caisse", "login, ajouter, logout, login, effacer"

2.3.4 Inspection continue

La différence 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 différents plugin. Les inspections ne remplacent pas les contrôles code manuelles, mais dans ces contrôles il y aura moins de défauts banale à traiter. Les objectifs des inspections sont engrené 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 (JavaNCSS, PMD [7]).
2. *Déterminer la dépendance*
   Les métriques de couplage (Afferent/Efferent Coupling) et l'instabilité d'un paquet de logiciel peuvent être des indications à comme important on paquet est. Les paquet qui sont utilisé très souvent il vaut mieux les tester très exactement (JDepend [8]).
3. *Imposer les standards de l'entreprise*
   Dans chaque entreprise il y a des règles comment il faut écrire le code. Des exemples très fréquemment sont que les variables n'ose pas avoir des noms trop courts et non-descriptive ou que les déclarations conditionnel doivent toujours être écrit avec des parenthèses (PMD).

7. (PMD, 2015)
8. (JDepend, 2015)

4. *Réduire le code copié*
   Le code source copié doit être avouer. Il y a des outils qui identifie des sections de code identique (PMD).
5. *Déterminer la couverture de code*
   La couverture de code par les testes est une métrique qui aide à déterminer quelles partie du code ont été négliger pendant écrire les testes (Cobertura [9]).

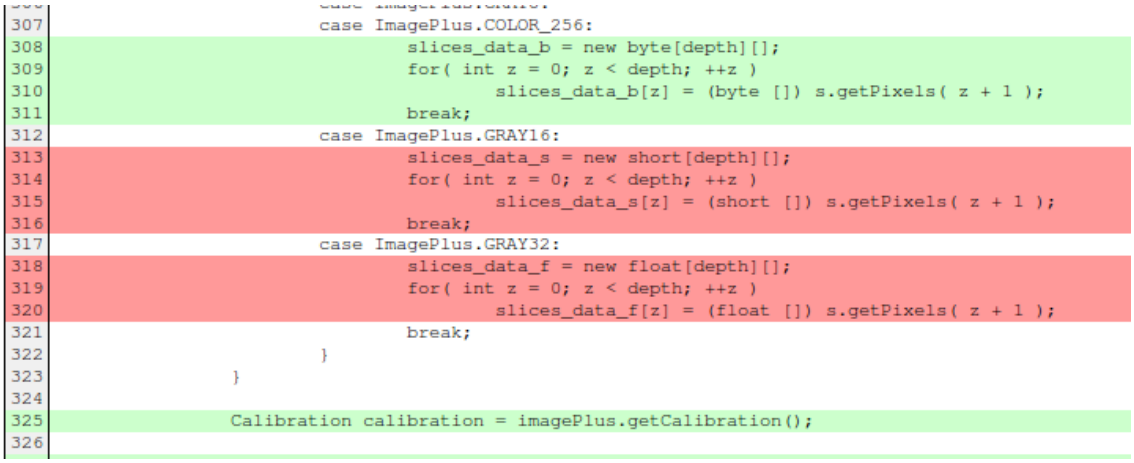

Figure 2.4 – Couverture de code exemplaire
fiji.sc (2015)

2.3.5 Information en retour continue

Pendant le processus de construction c'est indispensable de savoir ce que se passe. Chaque étape de construction doit fournir des informations à tout-le-monde affecté. Classiquement c'est réalisé par envoyer un courriel. Les possibilités plus modernes sont des plugins directement dans l'environnement de développement(Team Explorer for TFS intégré dans Visual Studio), dans une application de online-chat(TravisCI native, TeamCity, Jenkins avec plugins), comme notification sur l'ordiphone ou dans le système d exploitation(Native dans Windows 10 ou avec différents applications tierce partie)...

9. (Cobertura, 2015)

2.3.6 Déploiement continue

Le déploiement continue est un moyen pour délivrer la version actuelle aux utilisateurs immédiatement. Si un développeur fait un changement, la construction est initié automatiquement. Après ça les testes unitaires sont exécuté. Si tout les tests sont verts le développeur es informé et l'exécution des testes d'acceptation automatisées commence. Si la fonctionnalité du logiciel est assuré une nouvelle version est déployé immédiatement.

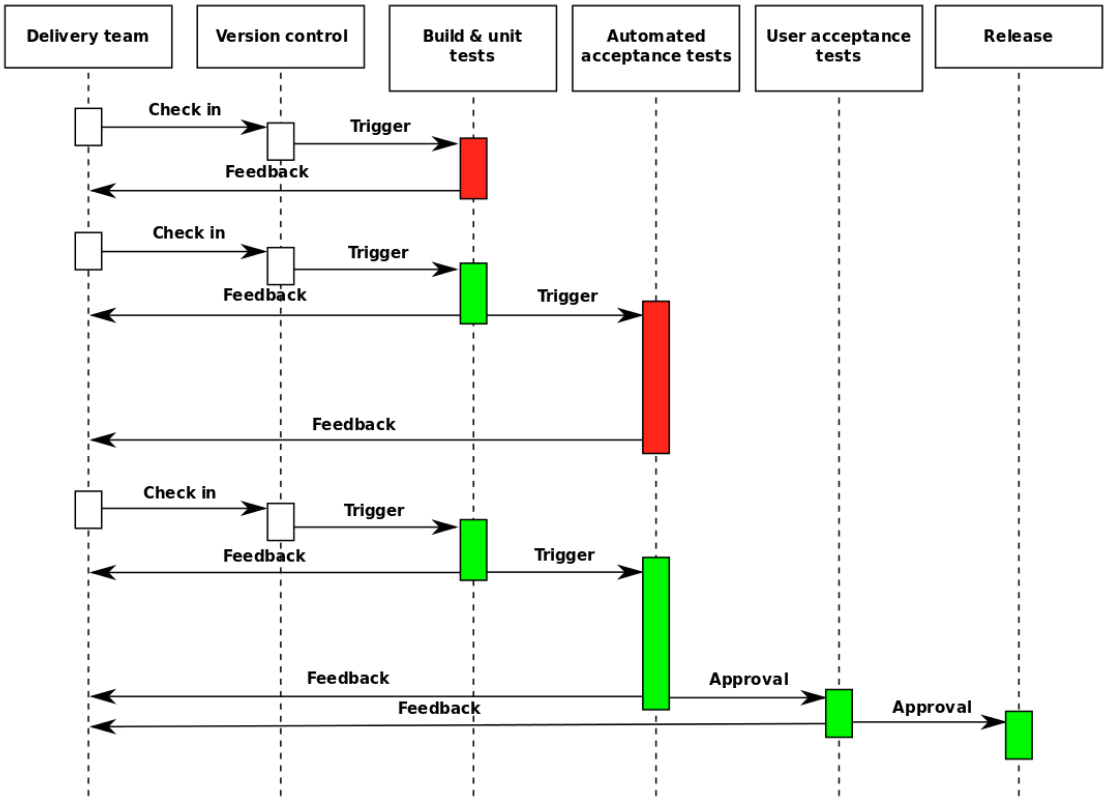

Figure 2.5 – Graphique déploiement continue
Humble (2015)

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 auras des problèmes et dans tous les logiciels il y auras 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 développeur 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). [10]

Logiciel pas près pour le déploiement

Si on fait l'intégration d'un système seulement à la fin du projet, 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 différentes sur la machine cible, ou même des dépendances qui n'ont pas été inclus pour le déploiement. Naturellement si le code source ne compile pas, le logiciel ne peut non plus être déroulé.

En commettre, construire et déployer 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 découvrit tôt. De plus il est aussi possible de déterminer 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 déployé pour les testes, les personnes testant le logiciels seront aussi automatiquement informé.

Il existe mêmes 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 nouvelles 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 différent ou le code redondant pourront devenir des erreurs dans le futur. Par exécutant des testes et des inspections régulièrement ces dérogations peuvent être trouvé 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. [11]

1. **Étendue de l'implémentation (Scope of implementation)**
   Avant de commencer l'implémentation d'un système de l'IC c'est absolument nécessaire de savoir de quelles composants on a besoin. Pas tous les projets nécessite 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 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-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 utilises des différents IDE. C'est pour ça que la construction doit être possible et fait à dehors d'une IDE.
7. **Réparer 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 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 réunissent 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

Pour tester les logiciels de construction nous avons compilé des projets de test de différent largeur sur les systèmes mesurant le temps.

3.1.1 Ant

Largeur	Temps(secondes)
Petit1	7.54
Petit2	8.01
Petit3	7.57
Grand1	27.42
Grand2	35.25
Grand3	28.17

3.1.2 Maven

Largeur	Temps(secondes)
Petit1	6.56
Petit2	7.10
Petit3	6.49
Grand1	24.32
Grand2	23.45
Grand3	25.01

3.1.3 Gradle

Largeur	Temps(secondes)
Petit1	3.26
Petit2	3.37
Petit3	3.41
Grand1	11.43
Grand2	12.25
Grand3	12.05

Conclusion

Gradle a des résultats meilleur que Ant et Maven, mais c'est plus compliqué a apprendre. Aussi la documentation de Gradle n'est pas très facile a comprendre parce-que elle est trop détaillé.

3.2 Serveur de l'intégration continue

3.2.1 Définition des critères

En dessous vous trouvez les critères avec une bref description qui seront utilisé pour évaluer les quatre serveurs de l'intégration continue qui ont été choisit pour ce travail. [1]

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 sélectionner 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 identifiant à des versions d'un logiciel ?
   — Extensibilité
   Est-il possible d'écrire des extension propre pour le serveur si nécessaire ?
2. **Générale**
   — Fiabilité et longévité
   — Environnement cible
   — Infrastructure
   — Couts
   — Type de logiciel
3. **Taille de la communauté**
   — Nombre d'utilisateurs
   — Nombre de plugins
4. **Utilisation**
   — Facilité d'utilisation
   — Complexité de l'installation

1. IBM (2006)

3.2.2 Apercu des resultats

Critères	Jenkins	TeamCity	Travis CI	Team Foundation Server
Caractéristique du produit				
Outils de gestion des versions	Subversion/CVS(+plugins)	Subversion/CVS(+plugins)	github/Git	Git/TFVC
Outils de construction		++ (CLI)	+	
Information en retour		+	++	
Labeling		✓	x	
Extensibilité	++	+	-	–
Générale				
Fiabilité et longévité	✓	✓	✓	✓
Environnement cible	tous	tous	Linux	Microsoft Windows
Infrastructure	On-premises	On-premises	On-premises/SaaS	On-premises/SaaS
Couts	gratuit	Freemium*	Freemium*	Freemium
Type de logiciel	Open Source (MIT)	Propriétaire	Open Source (MIT)	Propriétaire
Taille de la communauté				
Nombre d'utilisateurs	many	30'000 clients	240'000 projets	many
Nombre de plugins	++	+	x	-
Utilisation				
Facilité d'utilisation	many	+	++	many
Complexité de l'installation	many	+	++	many

Table 3.1 – Serveurs de l'IC

Freemium = C'est gratuit pour la version base, mais ca coute pour des editions plus grande (entreprise).
* gratuit pour des projets Open Source [2]

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

3.2.3 Jenkins

Caractéristique du produit Bien que Jenkins est écrit en Java il est capable de construire beaucoup des langues différents (PHP, Ruby, .Net avec des plugins et tout les autres lancent un script batch ou shellscript dépendent au système d'exploitation). Il y a une nouvelle version de Jenkins chaque semaine (utilisant IC avec soi-même trouvé sur https ://www.jenkins-ci.org/).

Figure 3.1 – Jenkins Logo

Générale Jenkins est un fork de l'outil Hudson qui était développe par Kohsuke Kawaguchi chez Sun Microsystems en 2008. Deux ans plus tard Kohsuke a eu des différences avec son nouveau employeur Oracle. Il a quitté et en 2011 présente la première version de Jenkins. Jenkins est un logiciel code source ouvert écrit en Java licenciée MIT.

Taille de la communauté Aujourd'hui Jenkins est installé sur 127000 machines[3]. Sur la site web officielle de Jenkins il y a des plugins à toutes fins.(1000++)

Utilisation Après lancer le service de Jenkins on peut ouvrir l'interface web sur 127.0.0.1 :8080. Là on peut définir des jobs de construction, gérer les droits d'accès, changer des paramètres et beaucoup plus. Au début le système gestion de version Git n'était pas inclue, seulement CVS et SVN mais aujourd'hui il y a un plugin. L'interface d'administration est très facile à utiliser.

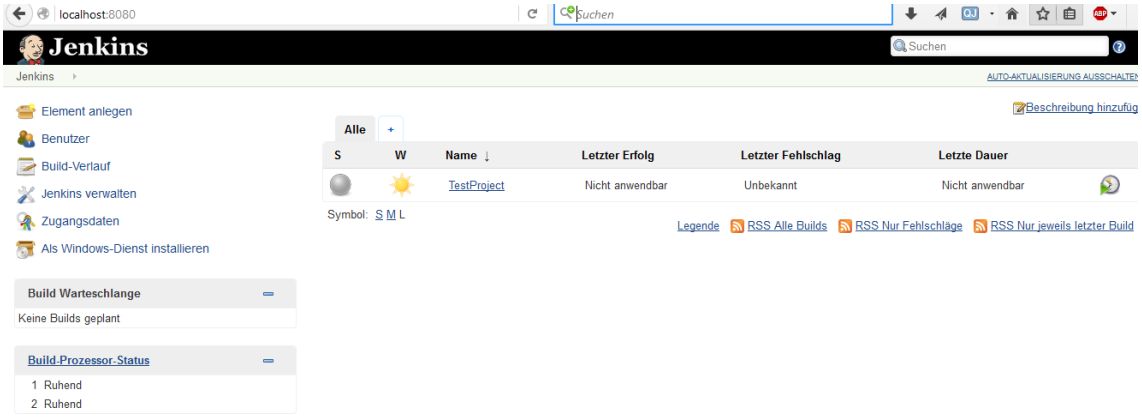

Figure 3.2 – Jenkins intérface d'administration

3. (Jenkins, 2015b)

3.2.4 TeamCity

Caractéristique du produit TeamCity est un serveur de l'intégration continue de JetBrains, les fabricants d'une grande nombre d'outil de développement (comme IntelliJ). Il est optimisé pour construire des projets de Java ou .NET, mais supporte aussi Python, Ruby et beaucoup d'autre langage avec des plugins. De plus il existe l'option de travailler avec la ligne de commande. TeamCity est très adaptable et peut être individualiser extensivement.
Tous les configurations et tous l'utilisation est effectué par l'interface web. Comme voies d'information en retour TeamCity supporte des emails, des messages Jabber ou directement dans l'IDE. De plus TeamCity supporte des "Build Tag" pour identifier des constructions. Si la fonctionnalité de TeamCity ne suffit pas, il est facilement possible d'écrire un plugin.

Figure 3.3 – TeamCity Logo

Générale L'infrastructure pour soutenir TeamCity doit être mis en place par l'entreprise. Il existe trois options de licence de TeamCity.

- TeamCity Professional (20 configurations et 3 agent de construction)
- TeamCity Enterprise (3-100 agent packs)
- TeamCity Additional Build Agent (+ 1 agent de construction)

La licence TeamCity Professional est gratuite mais limité. Pour les versions TeamCity Enterprise le nombre de projets est illimité, mais on peu incrémenter le nombre d'agent de construction pour atteindre une meilleure performance quand il y a beaucoup de processus de construction parallèle. Pour des projets Open Source TeamCity est gratuite, pour des jeunes pousses il y a des rabais. [4]

Taille de la communauté JetBrains affirme que 30'000+ clients utilise TeamCity pour exécuter l'intégration continue (Boeing, HP...). Il y a une grande nombre de plugins disponible sur le page web de TeamCity. [5]

Utilisation L'installation de TeamCity est assez facile. Il existe des archive des fichiers exprès pour des installations rapide. TeamCity est basé sur Java et utilise un serveur Tomcat. Pour l'installation rapide, tous ce qu'on doit faire est télécharger et extraire l'archive, et puis exécuter un script et compléter la configuration initiale. Pour un système productive une installation un peu plus complexe est prévu.
En utilisant TeamCity en premier on est confondu par la grande nombre d'options de configuration. Mais quand-même il était possible et pas très difficile de construire et mettre en rapport notre environnement teste. TeamCity est très volumineux, mais bien structuré et agréable pour l'utilisateur. Le faite qu'on peut faire toute la configuration sur l'interface web est un grande plus.

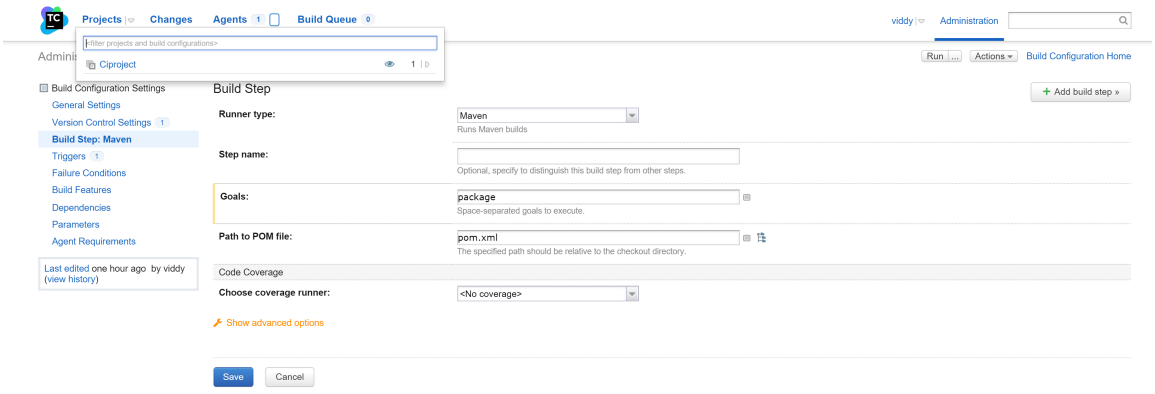

Figure 3.4 – TeamCity interface d'administration

4. (TeamCity, 2015a)
5. (TeamCity, 2015c)

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 [6]. 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. [7]. Chaque construction reçoit un identificateur numérique, 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 3.5 – 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 à disposition eux-mêmes (Linux).

Taille de la communauté Travis CI affirme sur la page web qu'il y a 246'506 projets Open Source qui sont testé et intégré sur leur plateforme. 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 exécuté automatiquement. Pour construire notre projet de test en java (maven), d'exécuter 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.

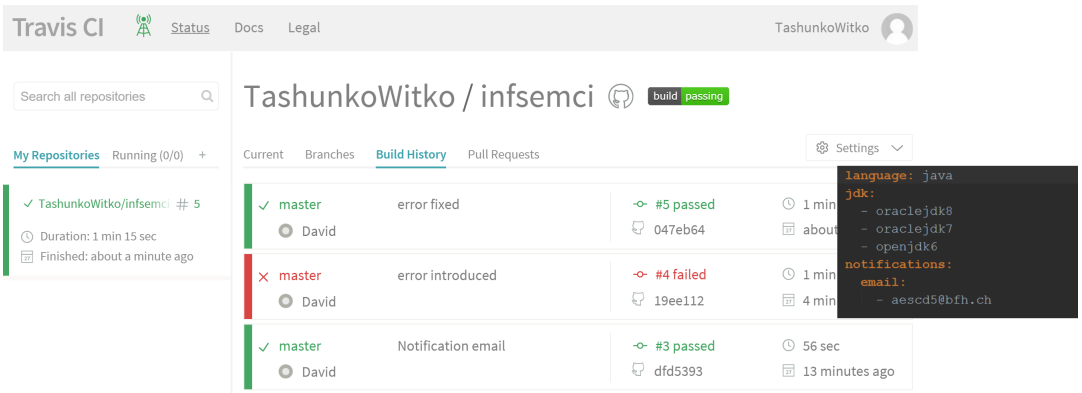

Figure 3.6 – Travis intérface d'administration et yml fichier

6. (TravisCI, 2015a)
7. (TravisCI, 2015b)

3.2.6 Team Foundation Server

Caractéristique du produit TFS est la forge logicielle de Microsoft. Le cycle de vie entier du logiciel agile est couvert. La gestion des versions (TFVC ou Git), des constructions, des résultats de test et beaucoup plus depuis on peut installer des plugins gratuit et payée facilement. Il y a une version "On-premises" qui est installé sur une machine/un serveur de l'entreprise mise a jour une foi par trimestre par Windows Update Services. Si on veut avoir la version la plus actuelle à n'importe quel temps il y a un service en ligne, fourni par Microsoft sur la nuage Azure.

Générale Le prix pour la version "On-premises" est calculé le même que pour la version SaaS listé en bas.

<5 personnes Gratuit, Temps de construction inclue 240 minutes/mois
6-10 personnes CHF 5.40/personne et mois
11-100 personnes CHF 7.20/personne et mois
101-1000 personnes CHF 3.60/personne et mois
>1001 personnes CHF 1.80/personne et mois
Agent de construction additionnel (hosted) CHF 36.10/agent
Agent de construction additionnel (locale) CHF 13.50/agent

Combiné avec une suscription MSDN beaucoup des services VisualStudio et Azure sont inclue.

Taille de la communauté Sur le nouveau magasin en ligne on peut télécharger nombreux plugins publié par Microsoft ou des développeurs indépendantes. Il y a des addons pour Visual Studio, VisualStudioCode et VisualStudio TeamServices gratuit et payé.

Utilisation L'interface utilisateur est très intuitive. Après l'enregistrement on peut lier un projet locale avec le projet team en ligne. Si on veut configurer la construction ou le déploiement automatisé il faut ouvrir l'interface web et suivre les instructions. La partie test est un peu confus au début, parce que les tests unitaires et les tests d'intégration sont inclue dans la construction et dans le chapitre test sont les tests d'acceptation recordé.

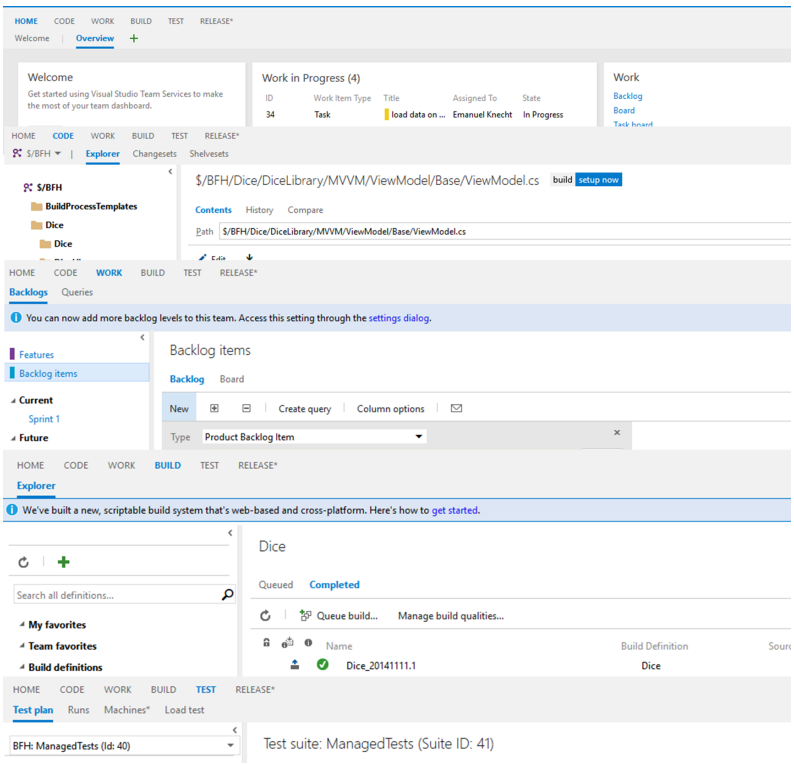

Figure 3.7 – Visualstudio Online interface d'administration

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.

fiji.sc. Coverage example, 2015. URL http://fiji.sc/_images/1/17/Coverage-file.png.

M. Fowler. Continuous integration, 2006. URL http://www.martinfowler.com/articles/continuousIntegration.html.

J. Humble. Continous delivery diagramm, 2015. URL https://en.wikipedia.org/wiki/Continuous_delivery.

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, 2015a. URL https://wiki.jenkins-ci.org/display/JENKINS/Plugins.

Jenkins. Jenkins statistics, 2015b. URL http://stats.jenkins-ci.org/jenkins-stats/svg/total-jenkins.svg.

R. Osherove. *The art of unit testing - Second Edition*. Manning, 2013. ISBN 978-1617290893.

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 buy, 2015a. URL https://www.jetbrains.com/teamcity/buy/.

TeamCity. Teamcity environment, 2015b. URL https://confluence.jetbrains.com/display/TCD9/Supported+Platforms+and+Environments.

TeamCity. Teamcity plugins, 2015c. URL https://confluence.jetbrains.com/display/TW/TeamCity+Plugins.

TFS. Tfs version control, 2015. URL https://msdn.microsoft.com/en-us/Library/vs/alm/code/overview.

TravisCI. Traviscidocs, 2015a. URL https://docs.travis-ci.com/user/getting-started.

TravisCI. Travis ci notification, 2015b. URL https://docs.travis-ci.com/user/notifications/.

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

Table des figures