



Intégration continues

Rapport 2 - Séminaire Info 2015

Filière d'études : Informatique

Auteurs : Emanuel Knecht, David Aeschlimann

Conseiller : Dr. Bernhard Anrig

Date : 11 janvier 2016

Table des matières

1	Introduction	2
1.1	Mission et Abstract	2
1.2	Approche	2
2	Intégration Continue	3
2.1	Histoire	3
2.2	Aperçu	4
2.2.1	Architecture exemplaire	5
2.3	Concepts de l'intégration Continue	6
2.3.1	Construction continue	6
2.3.2	Intégration continue de base de données	8
2.3.3	Tests continue	9
2.3.4	Inspection continue	10
2.3.5	Information en retour continue	10
2.3.6	Déploiement continue	11
3	Motivation et bénéfices	12
3.1	Éviter des risques	12
3.1.1	Logiciel pas près pour le déploiement	12
3.1.2	Découverte tarde des erreurs	12
3.1.3	Manque de visibilité du projet	12
3.1.4	Logiciel de basse qualité	12
3.2	Meilleures pratiques	13
4	Évaluation	14
4.1	Logiciels de construction	14
4.1.1	Java	14
4.2	Serveur de l'intégration continue	15
4.2.1	Définition des critères	15
4.2.2	Jenkins	17
4.2.3	TeamCity	18
4.2.4	Travis CI	19
4.2.5	Team Foundation Server	20
5	Conclusion	22
5.1	Bilan	22
	Bibliographie	23
	Table des figures	23

1 Introduction

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

1.1 Mission et Abstract

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 Intégration Continue et les concepts correspondants seront expliqués (Tests Continue, Intégration de base de données continue, etc). De plus les meilleures pratiques seront présentées et les bénéfices qu'on reçoit si on implémente les concepts et respecte les meilleures pratiques.

Dans la 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. Comme serveurs de l'intégration continue Jenkins, TeamCity, Travis et TeamFoundationServer ont été choisis.

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 administrateur. Après avoir défini 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 choisis et mieux les connaître, on a décidé d'installer, configurer et tester chacun. Pour ces tests on a créé des projets très simples en C++ et Java avec des tests unitaires. Ces projets ont été intégrés par les serveurs IC.

Après avoir fini 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 fois¹. Il parlait d'une intégration continue par des publications interne et chaque publication apporte l'application plus proche à la version finale.

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

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

Le premier livre publié sur la matière était "Continuous Integration"⁴ en 2007. Naturellement il y a beaucoup d'autres livres traitant des technologies ou outils concrets. Aujourd'hui le plus 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 diagramme d'un processus pareil.⁵

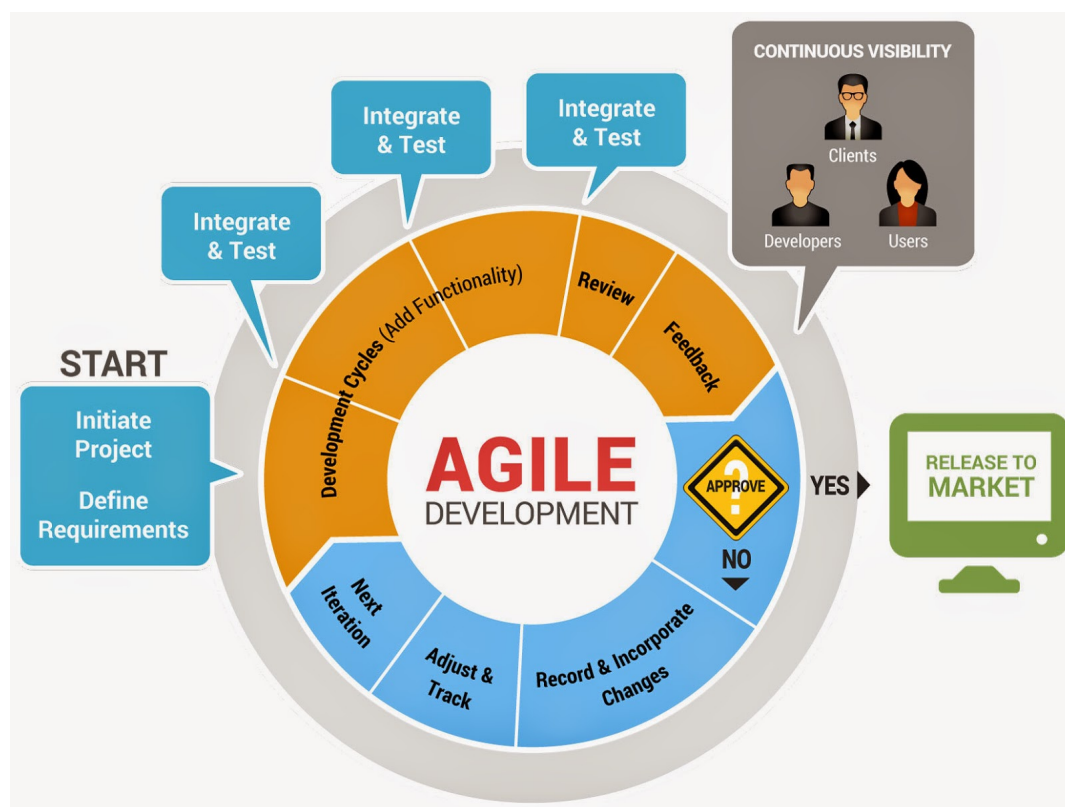


Figure 2.1 – Processus de développement logiciel

Les étapes récurrentes sont le développement, les tests, les contrôles, le déploiement, la réaction du client et après ça une nouvelle itération. L'idée principale de l'IC est d'automatiser des tâches des développeurs ou d'offrir des aides pendant tous ces étapes du processus de développement. Pendant presque tous les projets de développement on travaille en équipe. Tous les développeurs font des changements et ajoutent de la fonctionnalité chaque jour. C'est pour cela qu'il est nécessaire de réunir ces changements régulièrement et de vérifier si tous les composants marchent et coopèrent comme voulu (tests).

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 normalement considéré suffisante, naturellement le plus souvent le mieux.

De plus cette intégration doit être facile et automatisée, comme pousser un bouton. Après lancer le processus d'intégration tout le reste doit être contrôlé par le système IC. La définition et l'étendue de l'IC est ouverte, pas strictement limitée et fortement dépendant de l'application à développer. 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.

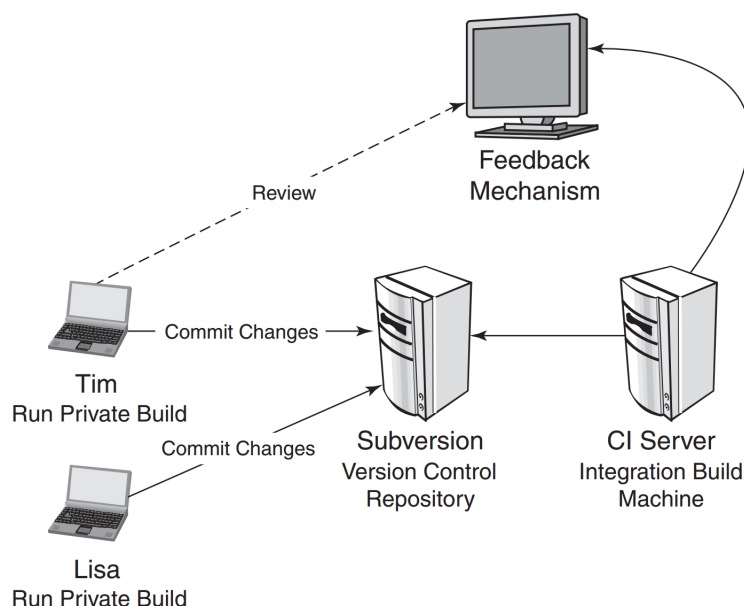


Figure 2.2 – Architecture exemplaire

Développement locale

Tous les développeurs travaillent 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 dédié, quelque fois 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 strictement plus de l'intégration Continue, mais aussi du **Déploiement et de la Livraison Continue** (Continuous Deployment, Continuous Delivery).

Information en retour

Après finir ce processus d'intégration les résultats des étapes sont 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

L'idée de la construction continue est de définir le processus de construction une fois, normalement dans un fichier de construction, qui est ensuite utilisé par le serveur d'IC de construire le projet à chaque intégration. Dans le fichier de construction les fichiers source et les dépendances sont définies. Tous les langages de programmation utilisent des outils de construction différents. Voici deux exemples.

Java

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

C/C++

Le premier logiciel de construction était make, qui existe depuis 1976 (Stuart Feldman, Bell Labs). Sur les plateformes basées sur Unix make est encore utilisé pour construire des exécutables. Pour construire une application avec make il faut fournir un fichier makefile qui contient les instructions de compilation. Supposons qu'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 de fichiers ça devient déroutant très vite. Par exemple si on a beaucoup de dépendances ou une application multi-plateforme c'est très difficile de ne pas oublier un élément.

C'est pour ça qu'il existe l'option de créer des makefile. Le makefile liste tous les fichiers et dépendances du projet.

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 l'introduction des variables dans le makefile.

Listing 2.6 – Makefile avec variables

```
CC=g++
CFLAGS=-c -Wall
SOURCES=main.cpp factorial.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 lisible. En 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 quels fichiers il faut compiler et quels sont les dépendances.

Listing 2.7 – CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)

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

add_executable(hello ${helloworld_SRC})
```

Tous ses étapes de construction peuvent être automatiser avec un serveur d'IC.

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.

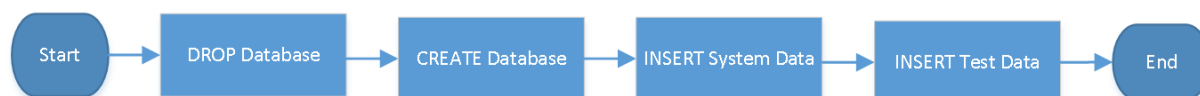


Figure 2.3 – Processus de l'intégration de base de données

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 plugin "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.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 optimalement tous les développeurs doivent avoir l'autorisation de changer la définition de 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 leur 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 Tests continue

Les tests existes pour contrôler la fonctionnalité d'un bout de code, d'un système ou d'une application complète. Le premier principe de base pour des tests automatisés est "Fail fast" (Échouer vite). On distingue trois types de tests différents. Comme règle d'or on peut dire que les tests unitaires prend des secondes, les tests d'intégration prend des minutes et les tests d'acceptation prend des heures.⁷

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

Après avoir tester les composant atomiquement, il est aussi nécessaire de tester l'interaction de ces composant avec des systèmes secondaire comme le système de fichier ou une base de données. Après ces tests d'intégration les erreurs logiques et d'intégration doivent être éliminé.

Tests d'acceptation

Les tests d'acceptation contrôle si les exigences du client sont respecté.

Il y a différents types de test d'acceptation :

1. *Tests d'interface utilisateur automatisé (code)*
Pour ces tests il faut spécifier des scénario. Une programme teste accède l'application et exécute le scénario avec des interactions avec l'interface d'utilisateur.
2. *Tests de performance*
Ces tests montrent quelles parties du code prennent la plupart des ressources.
3. *Tests d'exploitabilité*
Si le logiciel utilise une base de données on peut par exemple tester la vulnérabilité contre des attaques injection SQL.
4. *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"

Le concept tests continue décrit le fait, que les tests doivent être exécuter régulièrement ou mêmes continuellement. Le serveur d'IC est configuré de lancer les tests après la construction quand il y a un changement sur le dépôt centrale. Car les trois différent types de tests ont une durée différent, on se restreint normalement au tests unitaires pour des constructions normale. Les tests d'intégration ou d'acceptation sont exécuté par le serveur d'IC pendant la nuit ou le weekend. C'est pour cela qu'il faut avoir multiple configuration de construction et de tests par projet.

7. (Osherove, 2013)

2.3.4 Inspection continue

La différence entre les tests 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 plugins. 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 (Plugin : JavaNCSS, PMD⁸).

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 un paquet est. Les paquet qui sont utilisé très souvent il vaut mieux les tester très exactement et être prudent avec des changements (Plugin : JDepend⁹).

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 (Plugin : PMD).

4. *Réduire le code copié*

Le code source copié doit être avouer. Il y a des outils qui identifie des sections de code identique (Plugin : 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 (Plugin : Cobertura¹⁰).

```
307 case ImagePlus.COLOR_256:
308     slices_data_b = new byte[depth][];
309     for( int z = 0; z < depth; ++z )
310         slices_data_b[z] = (byte []) s.getPixels( z + 1 );
311     break;
312 case ImagePlus.GRAY16:
313     slices_data_s = new short[depth][];
314     for( int z = 0; z < depth; ++z )
315         slices_data_s[z] = (short []) s.getPixels( z + 1 );
316     break;
317 case ImagePlus.GRAY32:
318     slices_data_f = new float[depth][];
319     for( int z = 0; z < depth; ++z )
320         slices_data_f[z] = (float []) s.getPixels( z + 1 );
321     break;
322     }
323 }
324
325 Calibration calibration = imagePlus.getCalibration();
326
```

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

2.3.5 Information en retour continue

Pendant le processus de construction il est indispensable de savoir ce qui se passe. Chaque étape de construction doit fournir des informations à tous les personnes affecté. Classiquement c'est réalisé par envoyer un courriel aux personnes clés. Les possibilités plus modernes sont des plugins et notifications 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 message sur l'ordiphone ou dans le système d'exploitation (Native dans Windows 10 ou avec différents applications tierce partie).

8. (PMD, 2015)

9. (JDepend, 2015)

10. (Cobertura, 2015)

2.3.6 Déploiement continu

Le déploiement continu est un moyen pour délivrer la version actuelle d'une application aux utilisateurs immédiatement. Si un développeur fait un changement, la construction est initiée automatiquement. Après ça les tests unitaires sont exécutés. Si tout les tests complète avec succès le développeur est informé et l'exécution des tests d'acceptation automatisés commence. Si la fonctionnalité du logiciel est assuré une nouvelle version est déployé immédiatement. Il est très probable que ce processus est différent dans une autre entreprise ou applications. Les tests d'utilisateur ou des autre contrôles manuelles pourrait être exiger.

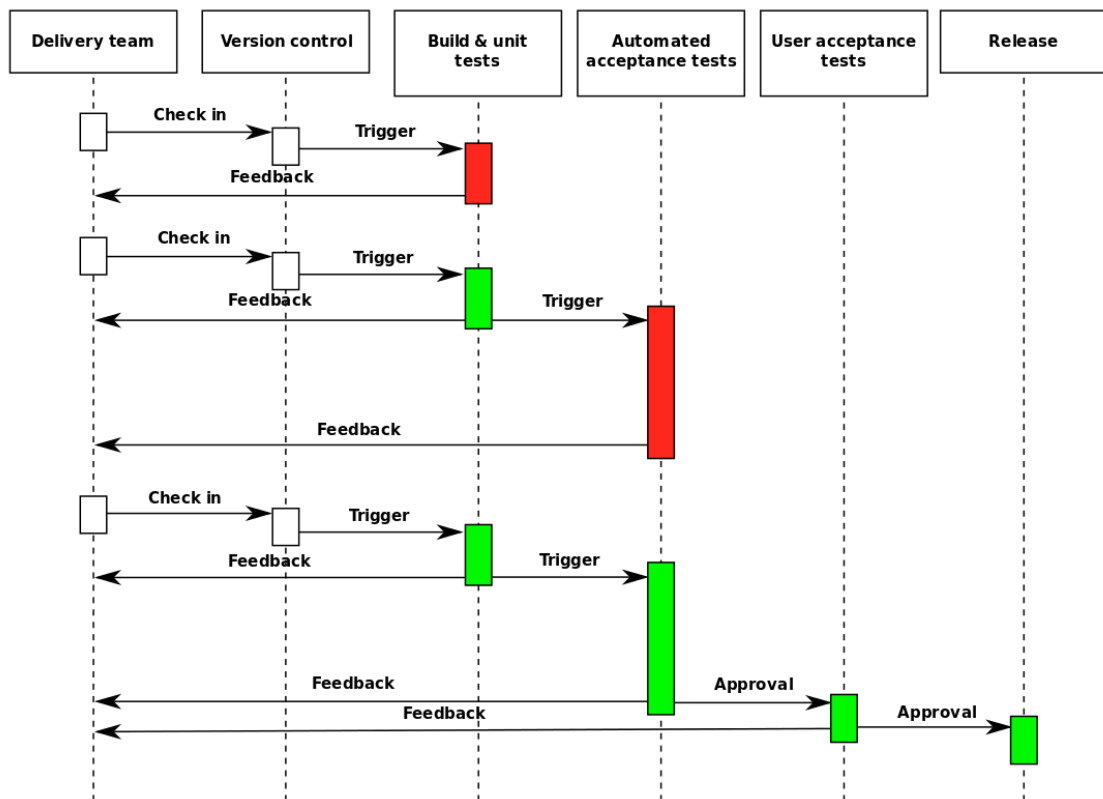


Figure 2.5 – Graphique déploiement continu
Humble (2015)

Un autre scénario très fréquent est qu'on a plusieurs environnements et serveurs pour déployer une application (local, development, testing, production). Un déploiement sur la plateforme development sera moins critiques qu'un déploiement sur la production.

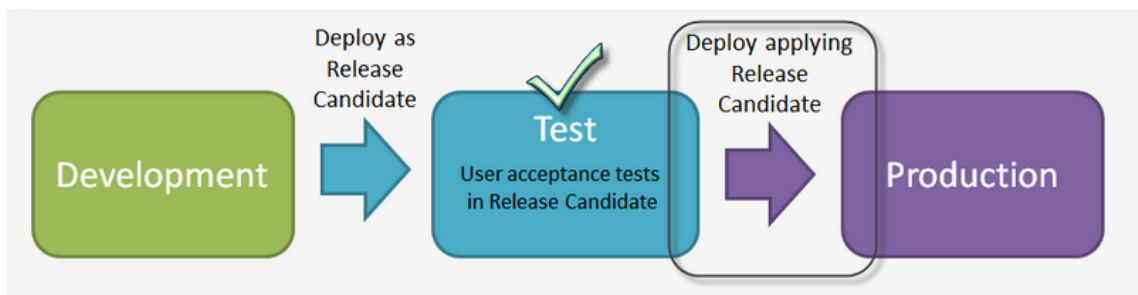


Figure 2.6 – Déploiement continu - Environnements
?

3 Motivation et bénéfices

La raison principale pour utiliser l'intégration continue 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 peuvent se concentrer sur ce qu'ils aiment faire le mieux, développer le logiciel.

3.1 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).¹

3.1.1 Logiciel pas prêts 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 un logiciel qui marche au moins partiellement.

3.1.2 Découverte tarde des erreurs

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

3.1.3 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. Le responsable de projet sait à chaque moment ce qui se passe avec le logiciel. Si il y a un problème avec les changements derniers toutes les personnes responsables seront contactées. Si une nouvelle version a été déployé pour les testes, les personnes testant seront aussi automatiquement informées.

Il existe mêmes des plugins qui font la visualisation du projet possible, en générant des diagrammes UML du code courant. Ça garantis une documentation du projet toujours actuel.

3.1.4 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. Comme ça la qualité et la facilité d'entretien du logiciel peut être augmenté.

1. (Duvall, 2007, p39)

3.2 Meilleures pratiques

En dessous vous trouvez quelques pratiques qui aident à optimiser l'efficacité d'un système d'intégration continue et donne des indications sur comment travailler avec un serveur d'IC.²

1. *Étendue de l'implémentation (Scope of implementation)*
Avant de commencer l'implémentation d'un système de l'IC il est absolument nécessaire de savoir de quelles composants on a besoin. Pas tous les projets nécessite les mêmes mesures, ça dépend 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. *Commencer le code souvent (Commit code frequently)*
Il est conseillé de commettre le code source au moins une fois par jour. On doit essayer de fragmenter le travail dans des morceaux petits et de commettre après chaque partie.
3. *Ne jamais commettre du code non-compilable (Don't commit broken code)*
4. *Éviter de télécharger du code non-fonctionnant (Avoid getting broken code)*
5. *Exécuter la construction et les testes localement (Run private builds)*
6. *Découpler le processus de construction du programme d'environnement intégré (Decouple build process from IDE)*
Le programme d'environnement intégré peut faire des pas dans le processus de construction qui ne sont pas transparent pour le développeur ou les développeur utilise des différents IDE. C'est pour ça que la construction doit être possible et être 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 réussissent pas on diminue la visibilité du projet.
10. *Garder les constructions vite (Keep builds fast)*

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

4 Évaluation

4.1 Logiciels de construction

4.1.1 Java

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

Largeur	Ant (secondes)	Maven (secondes)	Gradle (secondes)
Petit1	7.54	6.56	3.26
Petit2	8.01	7.10	3.37
Petit3	7.57	6.49	3.41
Grand1	27.42	24.32	11.43
Grand2	35.25	23.45	12.25
Grand3	28.17	25.01	12.05

Table 4.1 – Évaluation de logiciel de construction Java

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 très détaillé.

Comparaison de Syntaxe

Ema e churze verglich (bild zwüsche syntax war doch no toll

4.2 Serveur de l'intégration continue

4.2.1 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 choisis 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 d'une entreprise 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 ou des plugins pour le serveur si nécessaire ?

2. Générale

- Fiabilité et longévité
- Environnement cible
- Infrastructure
- Coûts
- 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)

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	EMA	✓✓(CLI)	✓	
Information en retour		✓	✓✓	
Labeling		✓	- -	
Extensibilité	✓✓	✓	-	- -
Générale				
Fiabilité et longévité	✓✓	✓✓	✓✓	✓✓
Environnement cible	multi-plateforme	multi-plateforme	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
Taille de la communauté				
Nombre d'utilisateurs		30'000 clients	240'000 projets	
Nombre de plugins	✓✓	✓	- -	-
Utilisation				
Facilité d'utilisation		✓	✓✓	
Complexité de l'installation		✓	✓✓	

Table 4.2 – Serveurs de l'IC

✓✓ = très bien | ✓ = bien | - = déficient | - - = pas existant/suffisant

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

On-premises = Les serveurs pour y installer le logiciel sont fournis par le client.

SaaS = Software as a Service

* gratuit pour des projets Open Source²

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

4.2.2 Jenkins

Caractéristique du produit Bien que Jenkins est écrit en Java il est capable de construire beaucoup des langues différents, comme PHP, Ruby, .Net avec des plugins. Tout les autres langage peuvent lancer 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/>).

Générale Jenkins est un fork de l'outil Hudson qui était développe par Kohsuke Kawauchi 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.



Figure 4.1 – Jenkins Logo

Taille de la communauté Aujourd'hui Jenkins est installé sur 127'000 machines³. 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 incluse, seulement CVS et SVN mais aujourd'hui il y a un plugin. L'interface d'administration est très facile à utiliser.

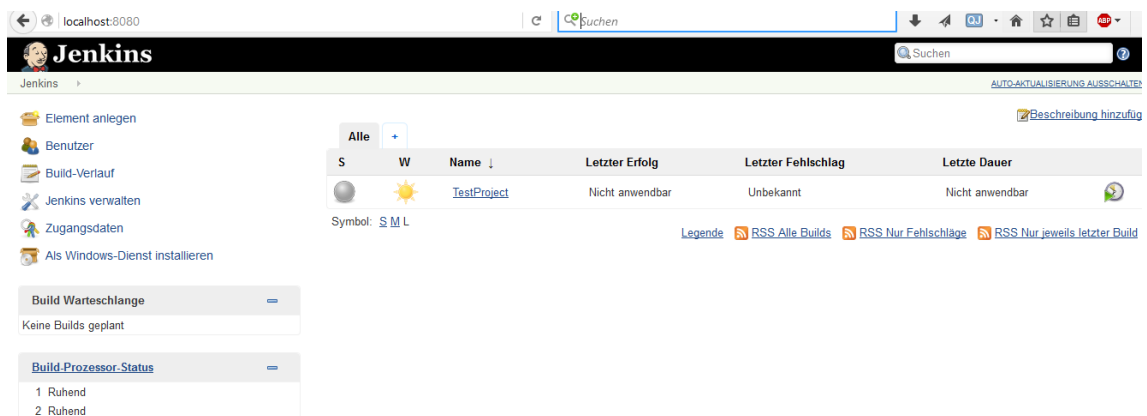


Figure 4.2 – Jenkins interface d'administration

3. (Jenkins, 2015b)

4.2.3 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 des notifications 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 4.3 – 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 payant (commençant de xxx) , 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.

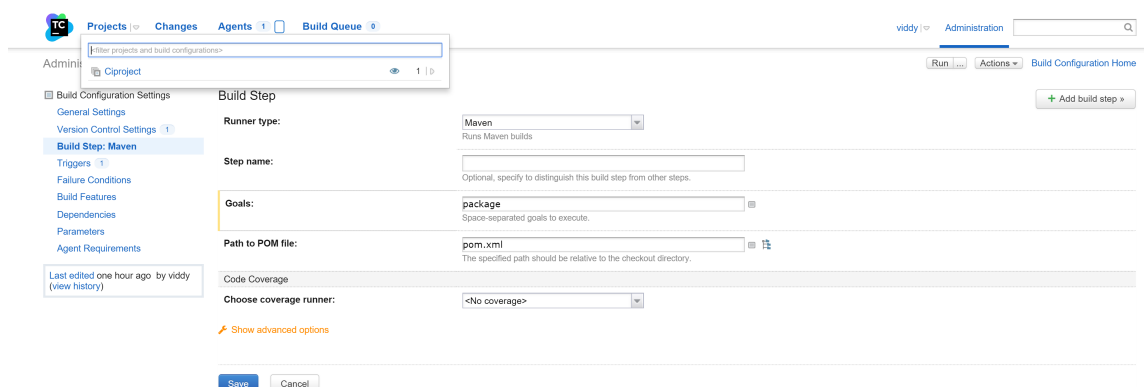


Figure 4.4 – TeamCity interface d'administration

4. (TeamCity, 2015a)

5. (TeamCity, 2015c)

4.2.4 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. Travis 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 4.5 – 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 est payant. 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 tests 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.

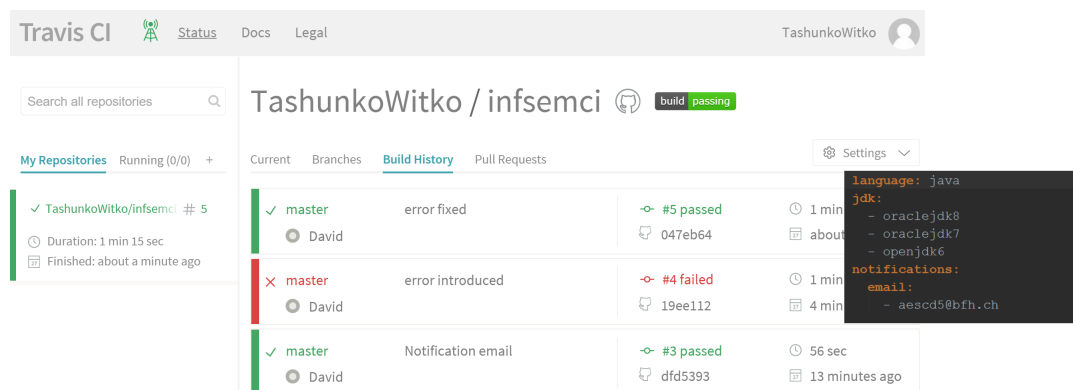


Figure 4.6 – Travis interface d'administration et yaml fichier

6. (TravisCI, 2015a)

7. (TravisCI, 2015b)

4.2.5 Team Foundation Server

Caractéristique du produit TFS est de la forge logicielle de Microsoft. Le cycle de vie entier du développement logiciel agile est couvert. La gestion des versions (TFVC ou Git), des constructions, des résultats de test et beaucoup plus car 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 fois 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 également pour la version SaaS listé en bas.

Taille	Couts
<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

Table 4.3 – TFS Couts

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

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 incluse dans la construction et dans le chapitre test sont les tests d'acceptation recordé.

EMAAA chasch uf ei site

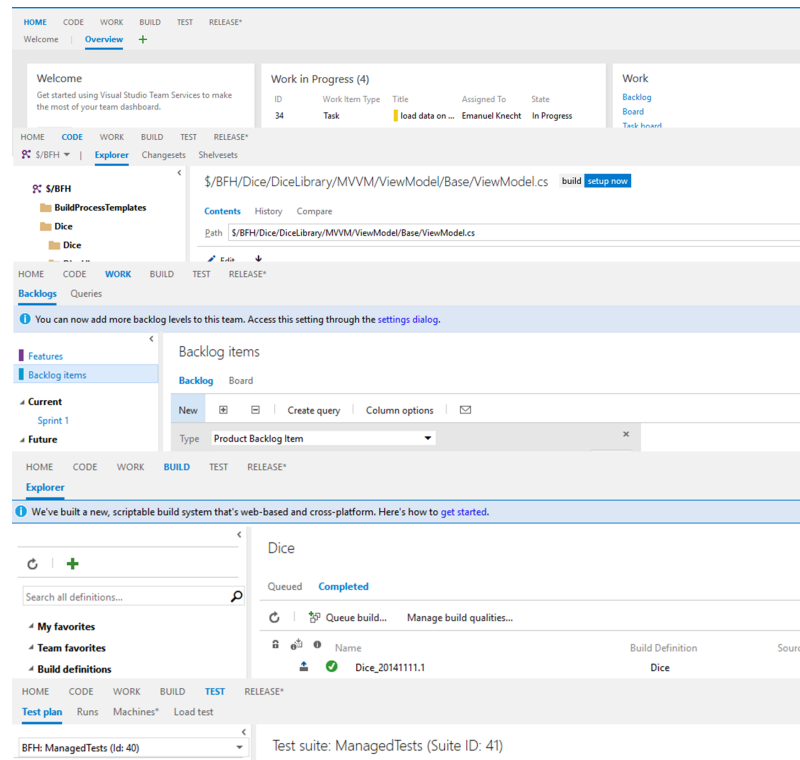


Figure 4.7 – Visualstudio Online interface d'administration

5 Conclusion

5.1 Bilan

EMAAA merci emaaaa xD

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. Oshero. *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. Travidocs, 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

2.1	Processus de développement logiciel	4
2.2	Architecture exemplaire	5
2.3	Processus de l'intégration de base de données	8
2.4	Couverture du code exemplaire	10
2.5	Graphique déploiement continue	11
2.6	Déploiement continue - Environnements	11
4.1	Jenkins Logo	17
4.2	Jenkins interface d'administration	17
4.3	TeamCity Logo	18
4.4	TeamCity interface d'administration	18
4.5	Travis CI Logo	19
4.6	Travis interface d'administration et yml fichier	19
4.7	Visualstudio Online interface d'administration	21