

Mémoire de fin d'études  
**Les enjeux de l'intégration continue**

**Maxime HORCHOLLE &  
Cédric TESNIERE**

Maître de Mémoire : **Joël GOY**

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>L'intégration continue</b>	<b>4</b>
2.1	Principe . . . . .	4
2.2	Qu'est-ce que c'est et pourquoi l'utiliser . . . . .	5
2.3	Les motivations des entreprises . . . . .	7
2.4	Les motivation au niveau projet . . . . .	8
<b>3</b>	<b>Les outils les plus utilisés du marché</b>	<b>9</b>
3.1	Tests . . . . .	9
3.1.1	Les tests unitaires . . . . .	9
3.1.2	Les tests fonctionnels . . . . .	13
3.1.3	Les tests d'intégrations . . . . .	18
3.2	Gestionnaire de versions . . . . .	18
3.3	Détecteur de copier coller . . . . .	19
3.4	Revue de code . . . . .	20
3.5	Analyseur de code . . . . .	20
3.6	Logiciel de suivi de problemes . . . . .	23
3.7	Test de couverture . . . . .	26
3.8	Coding style checker . . . . .	27
3.9	Serveur d'integration continue . . . . .	28
3.10	Conclusion . . . . .	31

---

<b>4</b>	<b>Les méthodes pour l'intégration continue</b>	<b>32</b>
4.1	Méthode agiles . . . . .	34
4.1.1	Kanban . . . . .	35
4.1.2	Scrum . . . . .	36
4.1.3	eXtreme Programming (XP) . . . . .	37
4.2	Le développement piloté par les tests (TDD) . . . . .	38
4.3	Behavior Driven Development (BDD) . . . . .	39
4.3.1	C'est quoi? . . . . .	39
4.3.2	Comment ça marche? . . . . .	40
4.3.3	Les solutions existante . . . . .	41
4.3.4	Exemple complet . . . . .	42
4.4	Conception pilotée par le domaine (DDD) . . . . .	43
4.4.1	Le principe du DDD . . . . .	43
<b>5</b>	<b>Les nouveautés</b>	<b>44</b>
5.1	Ça va bientôt arriver dans le cloud ! . . . . .	44
5.1.1	La livraison continue . . . . .	45
5.1.2	Intégration continue . . . . .	45
5.2	L'Intégration continue en local . . . . .	47
<b>6</b>	<b>Conclusion</b>	<b>48</b>
<b>7</b>	<b>Les sources</b>	<b>49</b>
7.1	Bug Trackers . . . . .	49
7.2	Analyseur de code . . . . .	49
7.3	Tests de GUI . . . . .	49

7.4	Tests d'intégration . . . . .	49
7.5	Tests fonctionnels . . . . .	50
7.6	BDD . . . . .	50
7.7	DDD . . . . .	50
7.8	Méthode agile . . . . .	50

# 1 Introduction

On entend de plus en plus parler d'intégration continue dans les médias spécialisés, de plus en plus d'entreprises se spécialisent dans ce domaine, mais le public sait-il réellement de quoi il s'agit. Sans doute, sait-il que cette technique sert à améliorer la qualité du code, mais en sait-il réellement plus.

C'est pour cela que nous avons décidé dans ce document de faire un panorama très exhaustif de ce qu'est l'intégration continue ainsi que de répondre à la question du pourquoi devrions nous choisir cette méthode et comment une équipe de développeurs en arrive à l'utiliser.

Pour répondre à notre sujet ce document sera découpé en trois grandes parties. Une première expliquera en quoi consiste globalement l'intégration continue et ce qu'elle peut apporter à un projet ou à une entreprise. Dans la seconde nous verrons les types d'outils les plus utilisés et pourquoi les mettre en place. Et enfin, dans notre dernière partie nous aborderont les outils encore peu connus et très peu utilisés par les entreprises qui pourraient bien révolutionner ce qui existe à l'heure actuelle en matière d'intégration continue et pour cela nous irons regarder ce qui se passe du côté du monde de l'open-source.

## 2 L'intégration continue

### 2.1 Principe

Pour expliquer ce qu'est l'intégration continue il faut s'attarder sur les causes de sa création. Avant qu'on invente le concept d'intégration continue, les projets se déroulaient en trois phases, la première consistait à s'entendre avec le client sur un certain nombre de fonctionnalités qu'il voulait implémenter dans son application, puis dans la deuxième phase les développeurs réalisaient une solution, on notera qu'en général que cette étape subie souvent des retards ou donne lieu à des applications fortement buggués. Bien évidemment cette phase se passe sans aucune communication avec le client. Puis arrive

la dernière phase ce que l'on appelle l'intégration, qui consiste à déployer et à tester la solution qui a été développée. Sur ce point bien souvent cela bloque, vu que l'équipe de développement n'a eu presque aucun contact avec le client, la solution peut ne plus correspondre à son besoin où ne pas être à son goût, dans ce cas le client sera mécontent. Il se peut que l'application soit inutilisable car pleine de bugs, car l'application n'a pas été correctement testée avant en conditions réelles (avec toutes les briques applicatives, dans un environnement similaire). Et bien souvent le code est d'une qualité extrêmement médiocre et donc immangeable et modifiable... Donc en général ce genre de projet fini à la poubelle bien rapidement.

C'est pour toutes ces raisons que l'on a créé l'intégration continue, si on devait résumer le pourquoi du comment en une phrase on pourrait dire :

*Vous n'aimez pas les phases d'intégration? Alors intégrez plus souvent!*

Cette phrase peut sembler contradictoire mais elle est pleine de sens, en effet en intégrant plus souvent (une fois par semaine par exemple) il est plus facile de corriger le tir, qu'au dernier moment, quelques semaines avant la livraison finale.

## 2.2 Qu'est-ce que c'est et pourquoi l'utiliser

« L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée. Bien que le concept existât auparavant[réf. nécessaire], l'intégration continue se réfère généralement à la pratique de l'extreme programming. » (*Source : Wikipedia*)

Quand on parle d'intégration, il ne s'agit en aucun cas d'un outil magique comme bien souvent on peut en entendre parler en informatique, mais d'un concept agile. La mise en oeuvre de ce concept se fait par l'ajout d'un certain nombre d'outils, que nous verrons plus tard dans ce document, chacun de ces outils a pour but d'améliorer la qualité globale du code qui est produit. L'intégration continue fait partie des 12 méthodes

d'eXtrême Programming (XP). Il existe plusieurs facteurs qui influent sur la qualité du code :

- Maintenabilité
- Testabilité
- Rapidité

Sur chacun des outils qui sont utilisés en intégration continue influe sur un de ces facteurs. Le but étant bien sûr que la qualité du code soit la meilleure possible afin de faciliter les prochaines relectures du code pour implémenter de nouvelle fonctionnalité et autres correctifs de bogue afin que le client soit content et n'est pas de mauvaise surprise à la fin, car un client heureux est un client qui fera sûrement appel à vous dans le futur.

L'intégration continue possède plusieurs avantages qui en résultent et c'est principalement grâce à ces avantages que l'on utilisera ce concept agile. Les avantages d'une intégration continue sont par exemple :

- Réduire les coûts pour la résolution des bogues, généralement un bogue est lié à la répercussion de la modification du code existant pour une nouvelle fonctionnalité lorsque l'on n'utilise pas différents tests automatisés à chaque build. La correction de ces bogues qui en résulte a un coût sachant que les bogues en recette pour le client coute plus cher qu'en développement.
- Meilleure réactivité en cas de bogue, si un développeur oublie de commiter un fichier, les tests échouera et permettra d'avertir le développeur de son erreur.
- Par l'intermédiaire aux différents rapports et indicateurs générés par des outils de reporting comme Jenkins, l'équipe en charge du développement ainsi que leur chef de projet voit en temps réel l'avancement et la santé du projet, le chef d'équipe pourra mieux définir les besoins et appliquera son budget selon les différents rapports reçus par mail.
- Les différents problèmes liés à l'intégration du logiciel sur un serveur seront détectés et réparés de façon continue, cela permet d'offrir des recettes plus rapides au client.

Bien qu'il possède certains avantages, l'intégration continue possède aussi certains inconvénients bien qu'en soit l'intégration continue ne possède pas d'inconvénient pour le mettre en place :

- Tous les bugs ne sont pas identifiés si la couverture des tests est insuffisante (voir la partie Coverage)

Le temps nécessaire à intégrer l'intégration continue est variable sachant que la durée de mise en place sera considérablement réduite à chaque projet. La première chose qu'une entreprise doit faire est de le mettre en place petit à petit afin d'habituer les développeurs à ce concept.

## 2.3 Les motivations des entreprises

Il faut en général déterminer avant de commencer si le projet utilisera l'intégration continue ou non, ce choix induirait des conséquences (positives ou négatives). Donc pourquoi de nos jours les entreprises optent de plus en plus pour l'intégration continue ? Dans le cas où certaine entreprise resterait perplexe à cette pratique car ils ne l'ont généralement jamais testé et pour cause, les entreprises pensent à la durée de l'installation des outils de l'intégration continue et plus particulièrement aux formations de ses développeurs ce qui reste un coût mais généralement les entreprises utilisent seulement la méthode SCRUM afin d'accélérer leurs développements avec plusieurs phases de déploiement par semaine. Mais sachant que l'informatique avance à grands pas chaque jour, les développeurs et le chef de projet prennent conscience des avantages de l'intégration continue et commencent à l'intégrer petit à petit sur leur projet et de conclure sur l'utilisation ponctuelle de cette pratique sur chacun de leurs futurs projets.

Principalement les points qu'ils favorisent la décision d'utiliser l'affiliation continue au sein d'une entreprise sont principalement les suivants :

- L'utilisation de l'intégration continue permet d'avoir des demandes de démonstrations non planifiées. Le projet étant constamment compilé et envoyé sur le serveur



de développement, cela permet au client de visualiser le rendu du projet à chacun des builds. Le projet est donc gérés par tranches et par lots conditionnels.

- Dans une entreprise des besoins varient continuellement en fonction des produits des concurrents éventuels et des changements légaux, réglementaires (contraintes d'importation, de confidentialité, etc.).
- Besoin d'intégrer rapidement les évolutions d'un projet en continu.

## 2.4 Les motivation au niveau projet

L'utilisation de cette méthode permet de réduire la durée, l'effort et la douleur provoquée par chaque intégration, de même, quand on réduit la durée entre chacune de ces intégrations, moins elles sont difficiles à effectuer. À propos des tests unitaires, l'intégration continue permet d'augmenter la vitesse des tests d'intégration et permet de détecter les problèmes et les défauts et donc de réduire les risques d'intégration.

L'intégration permet de résoudre dès le début du projet les problèmes liés au déploiement sur un serveur au lieu d'être confronté à ses erreurs à la fin du projet ainsi cela améliore la qualité des livrables, avec des outils d'analyse de code qui permet de réduire sa complexité (meilleure maintenabilité) cette pratique est en adéquation avec les bests practice de l'intégration continue. En utilisant un système d'intégration continue, l'équipe de développeurs doit forcément utiliser un gestionnaire de sources comme Git ou SVN qui permet d'avoir une meilleure traçabilité des changements et des déploiements. Ainsi en se focalisant sur le métier et non pas sur la technique, l'équipe gagne en productivité.

En utilisant cette pratique on utilise principalement les Principes « agiles » suivants :

- Fabriquer souvent
- Tester souvent (tests unitaires)
- Tester les performances souvent
- Intégrer souvent dans le SI

## 3 Les outils les plus utilisés du marché

### 3.1 Tests

On ne sait pas faire de logiciel sans défaut et le coût des corrections de bug peut coûter cher et prendre beaucoup de temps sur certains projets qui sont mal testés. Dans cette partie nous allons nous attarder sur les moyens de tester son application de manière automatique. Comme bien souvent il est totalement impensable (plus financièrement que techniquement) de tester à 100% un logiciel manuellement à chaque mise à jour du code. C'est pour cela qu'un certain nombre de technologies ont été créés afin de tester automatiquement si le code fonctionne toujours après des modifications.

#### 3.1.1 Les tests unitaires

C'est le test le plus répandu, mais aussi le plus efficace, car en général il est assez simple à mettre en place et couvre une très grande partie des besoins de test d'une application. Le test unitaire fait partie des tests "boîte blanche", ce qui signifie que le test a accès au code. Le test unitaire se présente sous la forme de classe objet contenant des méthodes visant à tester le code du projet. En général les méthodes sont écrites dans le même langage que le projet et utilisent des bibliothèques dédiées au test unitaire, en général on en trouve dans tous les langages elles se nomment généralement xUnit (où le **x** est la première lettre du nom du langage), par exemple JUnit pour Java, PUnit pour Python ... Dans ces méthodes pour tester du code on utilise ce que l'on appelle des **assertions**, il s'agit en fait d'une fonction du framework de test unitaire qui va tester la valeur attendue et la valeur actuelle si les deux valeurs sont différentes le test va alors passer en erreur, et le développeur saura donc que son code ne fonctionne pas, dans certaine méthode de développement que nous allons voir plus tard il arrive qu'on écrive même le code du test unitaire avant même d'écrire le code de la fonction que l'on teste (Test Driven Development). Le but ultime doit être de tester le maximum de comportement possible grâce aux tests unitaires afin de ne pas avoir de mauvaises surprises lorsqu'il y a modification du code existant.

Voici un exemple de classe de test unitaire en Java:

```
package fr.esgi.test.junit;

import junit.framework.TestCase;
import fr.esgi.java.Personne;

public class PersonneTest extends TestCase
{

    /**
     * L'objet de la classe que l'on veut tester
     */
    private Personne personne;

    /**
     * Test simple pour savoir si le constructeur ne plante pas
     */
    public PersonneTest(String name)
    {
        super(name);
    }

    /**
     * Ce déclenche avant chaque methode de test
     */
    protected void setUp() throws Exception
    {
        super.setUp();
        personne = new Personne("nom1", "prenom1");
    }
}
```

```
/**
 * Ce déclenche après chaque fin de méthode de test
 */
protected void tearDown() throws Exception
{
    super.tearDown();
    personne = null;
}

/**
 * Dans ce tests on utilise la méthode assertNotNull
 * du framework de test unitaire pour tester si l'objet
 * n'est pas null
 */
public void testPersonne()
{
    assertNotNull("L'instance est créée", personne);
}

/**
 * Ici on test si qu'il est impossible que le salaire
 * de la personne soit négatif et donc que la méthode
 * renvoie bien une Exception
 */
@Test(expected=IllegalArgumentException.class)
public void testException()
{
    this.personne.setSalaire(-1);
}
```

```
/**
 * On utilise la method assertEquals pour tester l'égalité
 * de deux chaines
 */
public void testGetNom()
{
    assertEquals("Est ce que nom est correct", "nom1", personne.getNom());
}

/**
 * On utilise la method assertEquals pour tester l'égalité
 * de deux chaines
 */
public void testSetNom()
{
    personne.setNom("nom2");
    assertEquals("Est ce que nom est correct", "nom2", personne.getNom());
}

}
```

Remarque à propos de l'exemple ci-dessus:

- toutes les méthodes de test sont préfixé par **assert**
- ses méthodes du framework facilite l'écriture de tests unitaire
- ici nous avons écrit uniquement des méthodes de test pour des assesseurs, mais il est possible tester des méthodes bien plus complexes.

**Les mocks tests** Pour tester les fonctions qui renvoie des résultat incertain selon la configuration (outils non installables, pas de connexion internet), il est possible d'écrire des mocks qui simule ce que doit renvoyer la fonction.

Voici un exemple concret: Dans notre cas nous voulons tester une fonction qui traite ce que retourne un outil en ligne de commande nous voudrions tester que si cet outil n'est pas installé qu'elle nous renvoie une erreur, et si cet outil est installé que ce que nous renvoie cette fonction est bien traitable par notre programme. Sans mock test il n'est pas possible de tester efficacement ce que nous renvoie cette fonction dans les deux cas. La façon de faire la plus simple est donc de créer un objet "mock" qui va simuler le comportement de cette fonction afin de tester que le reste du programme fonctionne. Le mock test n'est évidemment pas à utiliser n'importe où, en effet il ne sert qu'à remplacer le comportement de code sur lequel il ne nous est pas possible d'agir.

### 3.1.2 Les tests fonctionnels

Quoi de plus rébarbatif que de tester le bon fonctionnement de l'interface graphique de son application web, en effet il est souvent long et difficile de tester si tous les boutons renvoient bien sur la bonne page et sont bien disposés. Pour remédier à ce problème certain on a pensé à une solution simple pour automatiser ce type de test. Cette solution consiste à lancer un robot qui va effectuer toutes les tâches qu'un être humain aurait dû faire pour tester l'interface. Mais dès que le robot exécutera n'arrivera pas à reproduire l'action demandée on pourra en déduire que l'interface n'est pas correcte. Le gros problème de ce genre d'outils c'est qu'il ne permet pas de contrôler le design mais uniquement le bon placement dans l'interface des composants.

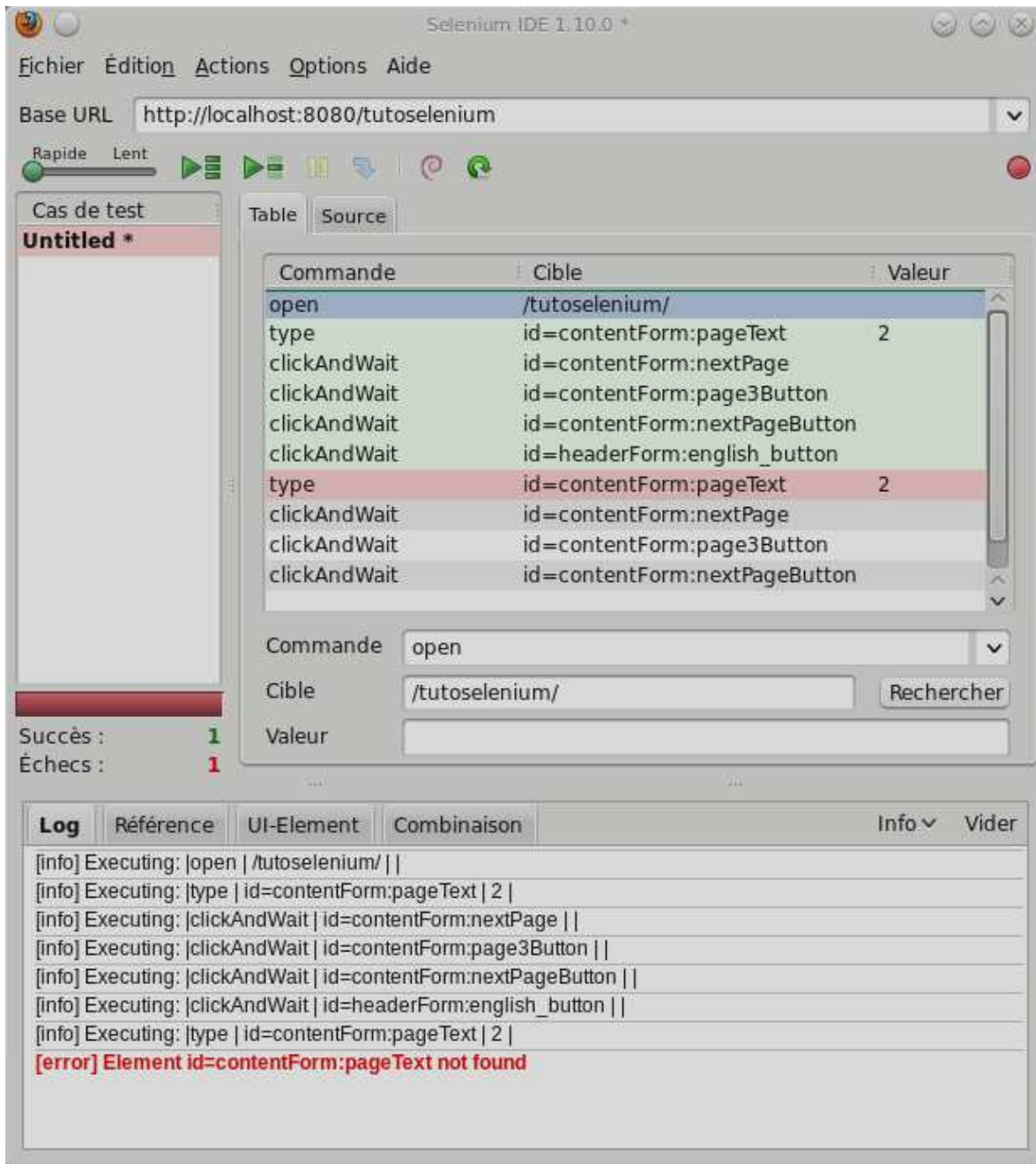
Il existe deux types de d'outils pour automatiser ces tâches. Les premiers qui prennent la main sur toute la machine qui sont pratiques si l'on veut tester des applications autres que web mais qui sont moins poussées en termes de tests et nécessitent l'utilisation d'une machine dédiée. Et d'autres qui ne simulent que l'utilisation d'un navigateur web qui s'intègre qui peuvent même parfois être directement pilotés depuis les tests fonctionnels de l'application comme c'est le cas pour Selenium.

Voici un exemple d'outil qui prend en charge toute la machine: Sikuli permet via son interface d'écrire des scripts qui se lancent sur la machine afin de reproduire le comportement que décrit dans le script. Dans l'image ci-dessous voici l'exemple d'une connexion à Gmail avec Firefox.



Voici un exemple d'outil qui permet d'automatiser les tests de GUI d'interface web sur différent navigateur via des scripts.

Nous avons choisi Selenium car il s'agit de l'outil le plus connu et le plus efficace qui permet celà. Dans l'image si dessous on voit le petit outil qui permet de créer un petit script que l'ont lance dans les tests fonctionnels de l'application.



Avec l'outil ci-dessus il est possible d'écrire des classes de tests fonctionnelles qui utilisent Selenium. Voici un exemple de code qui pourrait être généré:

```

public class Selenium {
    private WebDriver driver;
    private String baseUrl;
  
```



```
private boolean acceptNextAlert = true;
private StringBuffer verificationErrors = new StringBuffer();

@Before
public void setUp() throws Exception {
    // On instancie notre driver, et on configure notre temps d'attente
    driver = new FirefoxDriver();
    baseUrl = "http://localhost:8080/tutoselenium";
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}

@Test
public void testSelenium() throws Exception {
    // On se connecte au site
    driver.get(baseUrl + "/tutoselenium/");

    // On se rend page 1
    driver.findElement(By.id("contentForm:pageText")).clear();
    driver.findElement(By.id("contentForm:pageText")).sendKeys("2");
    driver.findElement(By.id("contentForm:nextPage")).click();

    // On est page 2, on va page 3
    driver.findElement(By.id("contentForm:page3Button")).click();

    // On sélectionne notre prochaine page dans la liste
    new Select(driver.findElement(By.id("contentForm:pageList_input"))).selectByVisibleText("3");
    driver.findElement(By.id("contentForm:nextPageButton")).click();

    // On est de retour page 1, on passe en anglais
    driver.findElement(By.id("headerForm:english_button")).click();
}
```

```
// Et on recommence le même enchainement
//...
}

@After
public void tearDown() throws Exception {
    driver.quit();
    String verificationErrorString = verificationErrors.toString();
    if (!"".equals(verificationErrorString)) {
        fail(verificationErrorString);
    }
}

private boolean isElementPresent(By by) {
    try {
        driver.findElement(by);
        return true;
    } catch (NoSuchElementException e) {
        return false;
    }
}

private String closeAlertAndGetItsText() {
    try {
        Alert alert = driver.switchTo().alert();
        if (acceptNextAlert) {
            alert.accept();
        } else {
            alert.dismiss();
        }
        return alert.getText();
    }
}
```

```
    } finally {  
        acceptNextAlert = true;  
    }  
}  
}
```

### 3.1.3 Les tests d'intégrations

Auparavant les tests d'intégrations devaient être effectués à la fin du développement et consistaient à tester manuellement que tous les “modules” de l'application fonctionnaient correctement ensemble. Ce genre de test quand on utilise l'intégration continue sont déplacés dans les tests unitaires et sont donc automatisés.

## 3.2 Gestionnaire de versions

Un gestionnaire de versions sert à stocker, versionner et partager son code. On utilise surtout des gestionnaires de versions quand le besoin de partager le code source entre les différents membres d'une équipe se fait ressentir. En effet, le plus souvent on stocke son code sur un gestionnaire de source alors que l'on est le seul utilisateur à peu d'intérêt, à part de celui de ne jamais perdre ses anciennes sources et ses évolutions. Les plus gros avantages se font ressentir sur les projets où il y a une équipe en effet, dont voici les principaux avantages d'avoir un gestionnaire de sources quand on est en équipe :

- Partager le code plus efficacement et ainsi éviter de partager son code avec dropbox ou par des emails.
- Tous les membres de l'équipe ont toujours une version du code à jour. Donc un gain de temps énorme lors de la mise à jour.
- Garder un historique de toutes les modifications et de qui les a effectuées. Ce qui permet un retour rapide en cas d'introduction de bug et de plus avoir peur de modifier le code source.

- Géré finement vos versions. Par exemple figer une version stable qui ne bougera plus et qui contient un nombre limité de bugs (une version “stable”).

Il faut savoir qu’il existe deux types de gestionnaire de sources, les centralisés qui existe depuis des années et les décentralisés qui inondent le marché depuis quelques années car il possède toutes les qualités des outils centralisés mais ajoute encore d’autre fonctionnalité dont voici les avantages de ces gestionnaires décentralisés face au centralisés.

- Possibilité de faire des commits atomiques sur sa machine et les pousser ensemble en une seule fois sur le serveur maître pour résoudre un bug ou créer une fonctionnalité. Il est aussi partie de travailler hors ligne ce qui est totalement impossible en centralisé.
- Un aspect communautaire renforcé :
  - Une jolie interface graphique où chacun peut voir et annotez-le commit de l’autre.
  - Chacun peut proposer des idées postées des bugs, ...
- Plus de sécurité :
  - Si les droits d’accès sont bien fait le développeur doivent passer par la case “pull request” c’est-à-dire qu’il faut que son commit soit accepté par l’administrateur pour être mis en ligne sur le serveur maître.

Au vu des fonctionnalités proposées par les serveurs de sources décentralisées il est conseillé d’opter pour ceux-ci, en particulier pour Git<sup>1</sup>.

### 3.3 Détecteur de copier coller

Rien qu’au titre on voit tout de suite l’intérêt de cet outil en effet il permet de vérifier que le code n’est pas remplis de copier-coller plus ou moins justifié. En effet la multiplication

---

<sup>1</sup>Plus d’info sur Git : <http://fr.wikipedia.org/wiki/Git> et <http://git-scm.com/>

des copié-coller rend le code de moins en moins maintenable car si un morceau de code copier-coller un peu partout dans le programme est modifié à un endroit car il entraîne des bugs il faudra mettre aussi à jour tous les autres morceaux où le code a été copié-coller. Ce qui rend donc le code très vite inmaintenable.

L'outil en lui même va aider les équipes à détecter les morceaux de code recopie ce qui permettra donc aux développeurs d'identifier les parties du code qu'il faut factoriser au plus vite pour rendre le code plus maintenable et éviter les modifications en chaîne de code, les bugs et régressions.

Exemple de détecteur de copier/coller : Mess Detector<sup>2</sup> pour Java

### 3.4 Revue de code

La revue de code consiste à faire relire le code par une personne qui n'a pas écrit ce code en presence ou non de la personne qui à codé. On voit tout de suite l'effet positif principal de cette technique: si une personne extérieur vérifie le code de l'autre il peut évaluer la qualité du code qui a été écrit et faire des retour à la personne qui l'a écrite. Il peut être aussi intéressant de faire participer la personne qui a codé à la relecture du code afin d'expliquer ce qu'il a fait à l'autre, cette méthode permet en plus de vérifier la qualité du code, de partager l'expérience et les connaissances de chacun pour éviter qu'une seule personne soit au courant d'un morceau de code spécial (work around, etc...)

### 3.5 Analyseur de code

Une analyse du code permet de générer des données d'analyse sur la mise en conformité par rapport aux standards du marché car la qualité d'une application est directement liée à la qualité du code et à la productivité.

De nombreux outils permettent de contrôler quelques aspects de cette qualité du code, principalement sur l'exécution de tests unitaires, l'analyse de la couverture du code par

---

<sup>2</sup>Mess Detector : [http://fr.wikipedia.org/wiki/PMD\\_\(logiciel\)](http://fr.wikipedia.org/wiki/PMD_(logiciel))

ces tests, la vérification du respect des règles de codage, etc. Ainsi ces outils permettent d'avoir une confiance accrue en son application ! Un contrôle fréquent de la qualité du code va donc pousser l'équipe de développement à adopter et à respecter certains standards de développement. Un code qui respecte ces standards est un code plus sûr car cela permet de trouver immédiatement les erreurs.

Lorsque l'on sait que le coût de la correction d'une erreur augmente considérablement avec le temps, un outil de surveillance permet la détection précoce de ces éventuels problèmes et l'on comprend très vite l'importance de la détection rapide des erreurs ...

Voici quelques exemples de resultat que pourrait renvoyer un analyseur de code.

Les deux images qui suivent sont extraite de l'analyse du projet PHP sur Scrutiniz-ci

**MhorCvToPdfExtension** A last analyzed about 4 hours ago

↳ Parent: [DependencyInjection](#)

Complexity	Size/Duplication	Test Coverage	Importance
Total Complexity <span style="color: green;">1</span>	Total Lines <span style="color: green;">11</span>	Coverage <span style="color: green;">100 %</span>	Changes <span style="color: green;">2</span>
Complexity/M <span style="color: green;">1</span>	Duplicated Lines <span style="color: green;">0 %</span>	Bugs <span style="color: green;">0</span>	Features <span style="color: green;">0</span>

+ 1 Method

Code Coverage Issues


```




1 <?php
15 class MhorCvToPdfExtension extends Extension
16 {
17     /**
18      * {@inheritdoc}
19      */
20     public function load(array $configs, ContainerBuilder $container)
25     }
26 
```


Dans l'image ci-dessus nous voyons l'analyse du code d'une classe. On y voit toutes les métriques (complexité, nombre de ligne, taux de couverture, nombre de ligne). Grâce à ces métriques une note est attribué à la classe.

Rating ▲	Name	Coupling	Cohesion	Size	Complexity	Test Coverage	Changes ○
A	GenerateCvCommand	?	?	38	4	79 %	4
A	DefaultController	?	?	3	0	100 %	2
A	Configuration	?	?	17	1	0 %	2
A	MhorCvToPdfExtension	?	?	11	1	100 %	2
A	CvGenerator	?	?	28	3	67 %	2
A	MhorCvToPdfBundle	?	?	3	0	100 %	1
A	Cv	?	?	113	8	0 %	1
A	Experience	?	?	149	10	0 %	1
A	Person	?	?	350	24	0 %	1
A	Skill	?	?	31	2	0 %	1
A	SkillType	?	?	31	2	0 %	1
A	AppKernel	?	?	38	4	?	2
A	GenerateCvCommandTest	?	?	42	2	?	6

Ici on voit une vue plus globale de tout le projet on y voit toutes les notes des classes du projet.


[Blog](#)
[Public Analyses](#)
[What we analyze](#)
[Pricing](#)
[Account](#)


**mhor / MhorFipPlaylistBundle #2**

7 days ago, duration: a few seconds

Edit project



15 minutes  
to get the Bronze Medal

**Severity**  
1 Critical  
1 Major  
13 Minor

**Category**

Violations (15)
Fixed
Ignored

► PHP debug statements found Critical Security ?

► exit() and die() functions should be avoided Major Bugrisk ?

► Unused method, property, variable or parameter 4 Minor Deadcode ?

► Unused use statement should be avoided 9 Minor Deadcode ?

Cette capture d'écran a été effectuée par un analyseur spécialisé dans le framework PHP Symfony2. On voit qu'il est aussi capable en plus de l'outil précédent de détecter des

erreurs lié au code et de donner des conseils afin de résoudre les problèmes.

### 3.6 Logiciel de suivi de problemes

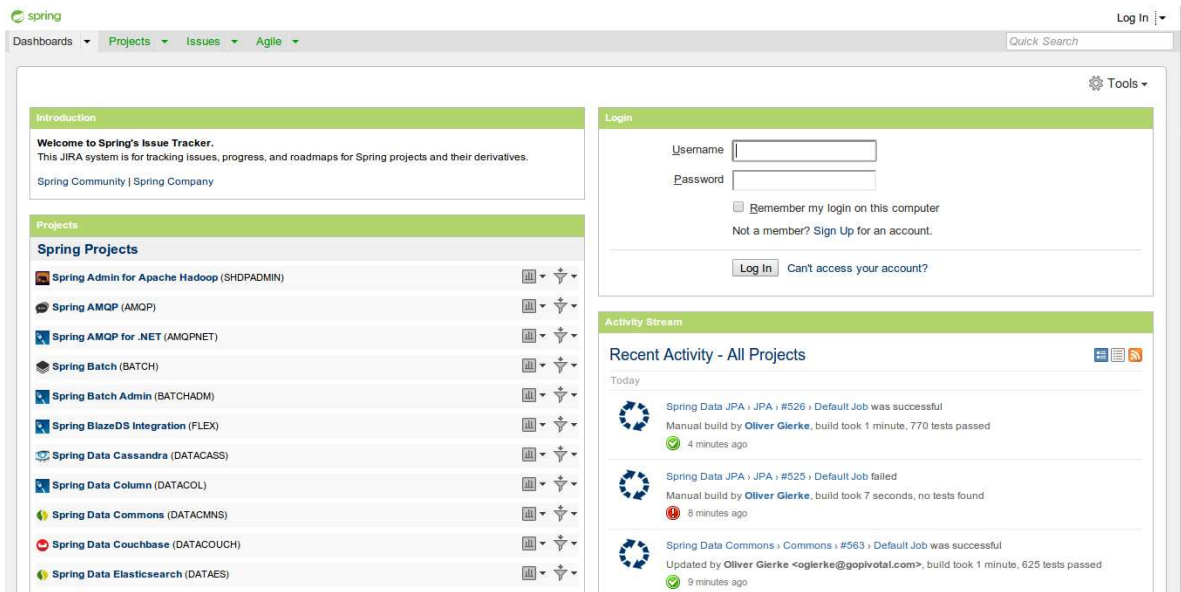
Aussi communément appelé Bug trackers se sont tous les outils qui servent à suivre l'évolution du projet. Dedans est consigné les évolutions et les bugs de l'application. Ce qui est très utile pour garder un historique de tout ce qui a été réalisé depuis le début. C'est aussi un outil indispensable pour distribuer les tâches.

Avant la création de ce genre d'outils la gestion de projets était bien plus difficile et l'équipe était beaucoup moins réactive puisque l'information était souvent soit non mises à jour où pire complétement indisponible. Avec un bug tracker efficace c'est un jeu d'enfant de rentrer des tâches, des bugs, de voir l'état d'avancement de ses collègues. Les bugs tracker moderne proposent aussi de nombreuses autres fonctionnalités comme des forums, des diagrammes de Gantt, etc.

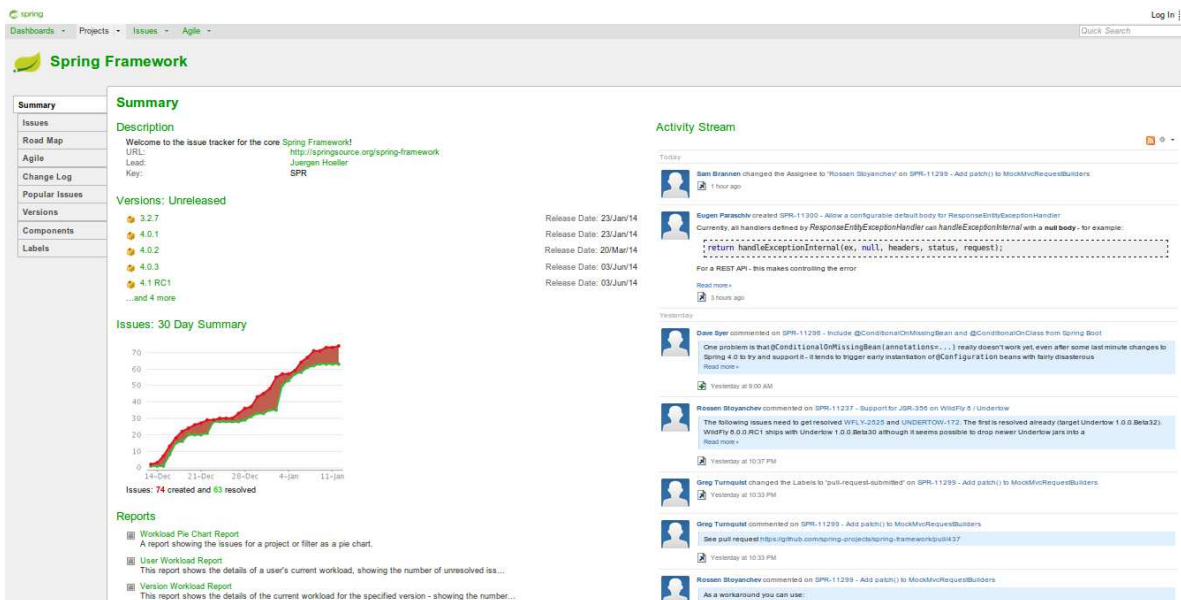
La majorité des bugs tracker peuvent/doivent être installés sur un de ses serveurs dans ce genre de solutions on trouve Redmine (gratuit) ou Jira (payant pour les projets non open-source) qui est de très bons bugs trackers pour les gros projets . Mais récemment un nouveau type de bugs trackers est apparu : les bugs tracker sur le cloud, ils sont souvent fait pour être utilisé avec les méthodes agiles et sont surtout là pour les projets de petite envergure (équipe restreinte, taille du projet limité). Ces solutions sont souvent gratuite de base et propose un plan payant pour les utilisateurs professionnels. Dans cette catégorie on trouve des sites comme Trello, Assana ou waffle.io.

Ci-dessous nous avons choisi de vous présenter deux type de logiciel, le premier est le plus connu et complet et s'adapte à toutes les situations mais est assez lourd à prendre en main, il s'agit de Jira.





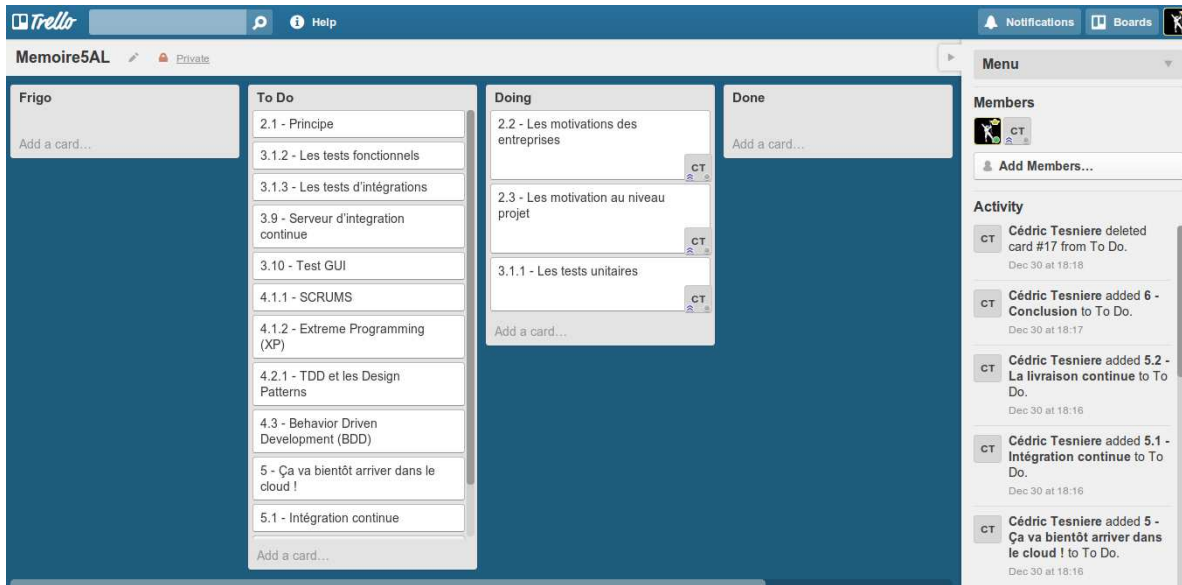
Ici on voit la page d'accueil avec tous les projets et les dernière activité de ceux-ci.



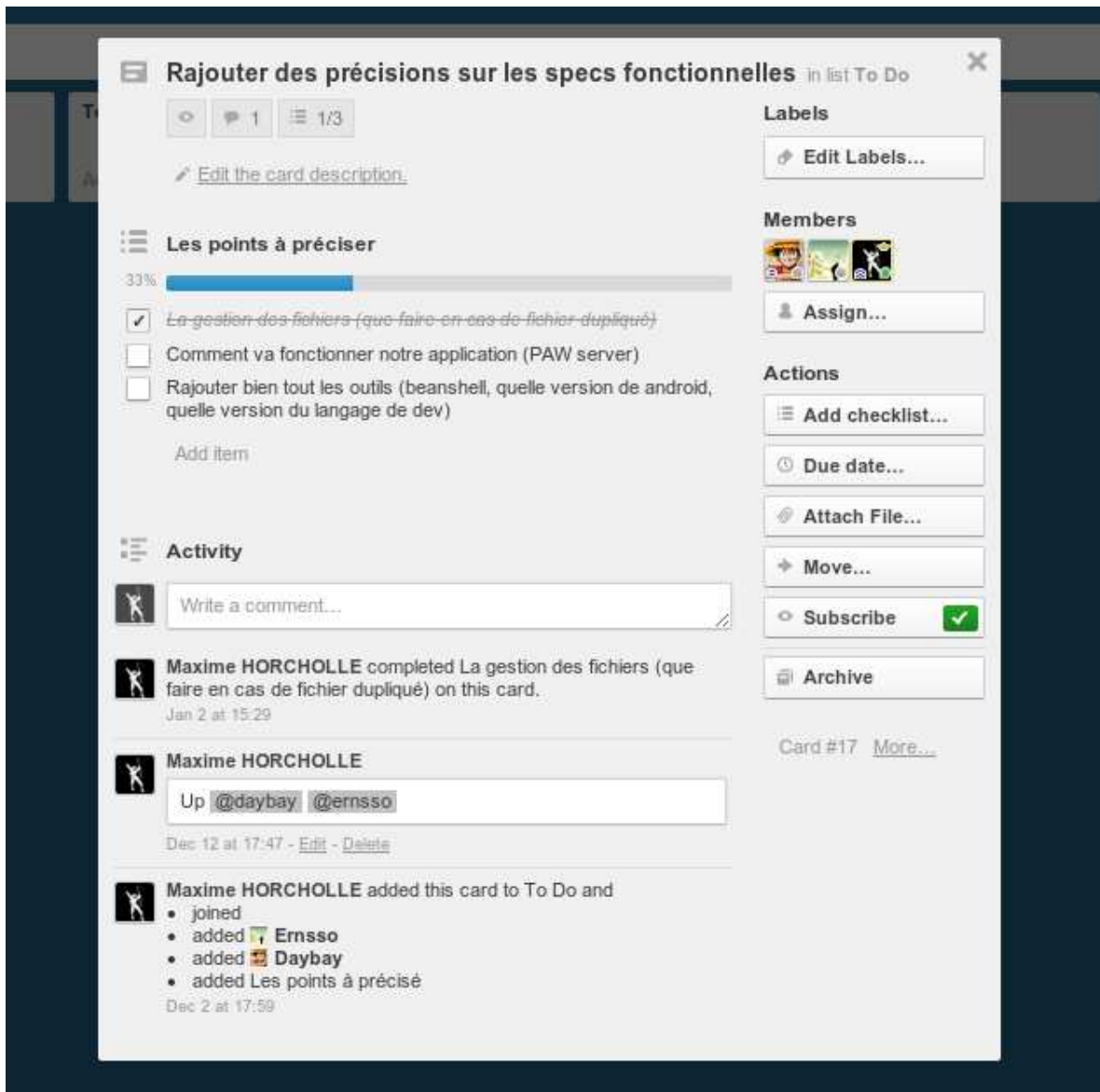
Lorsque que l'on s'intéresse plus particulièrement à un projet on dispose d'un certain nombre de d'information plus où moins intéressant.

Maintenant voyons Trello qui est un système plus léger bien plus adapté au développement Agile, en effet l'interface est fait pour suivre le concept de la plupart des méthodes agiles, meme si il faut reconnaitre que Jira sera tout de même plus adapté pour de plus grosse équipe (ce qui ne suis pas les préconisation des méthodes agiles) grâce a des plugins

dédiés



Ici on reconnait désormais fameux tableau de post-it cher à toutes les méthodes agiles celui-ci est personnalisable selon ses besoins.



Voici ce que l'on peut voir lorsqu'on veut plus d'information sur une "tâche"

### 3.7 Test de couverture

Après tout ce que nous avons pu voir j'espère que vous avez compris que le test est une chose essentielle en intégration continue. Mais une question se pose comment estime-t-on qu'un logiciel est suffisamment testé ? Une question bien difficile à répondre sauf dans un cas : celui des tests unitaires, en effet il existe des solutions dans tous les langages

qui calculent ce que l'on appelle le taux de couverture. Il s'agit du pourcentage de ligne qui est testé via les tests unitaires sur le nombre total de ligne du projet, donc plus on se rapproche de 100% plus les tests unitaires couvrent tous les cas possibles. Quelques solutions vont même plus loin dans le test de couverture en proposant par exemple de consulter le code du projet et de voir visuellement les lignes tester sont celles qui ne le sont pas encore. Cette fonctionnalité est aussi la bienvenue pour voir les parties des fonctions qui n'ont pas été testé, par exemple il est rare de voir les développeurs testés tous les cas d'erreur possible d'une fonction.

En général lorsqu'un logiciel est testé à moins de 50% on doit commencer à se poser des questions. Et recadrer au plus vite le projet afin de faire remonter le taux de couverture afin d'être sûr que le projet n'a pas subi de régressions. On estime un projet bien testé unitairement lorsqu'il a un taux de couverture supérieur à 70%. Et qu'il est très bien testé en 90% et 100%, le taux de 100% reste assez utopique pour les gros projets.

Pour finir sur le taux de couverture il faut juste signaler qu'il ne teste en aucun cas la qualité des tests unitaires donc si les tests ne sont pas pertinents (pas d'assertions, etc...) le logiciel ne le détectera pas...

### 3.8 Coding style checker

Cet outil fait partie des outils les plus importants pour faciliter la relecture et la maintenabilité du code. En effet, il permet de tester si les normes de codage fixé sont bien respecté par les programmeurs. Ces règles doivent être respectées car si elles ne sont pas suivi à la lettre on se retrouve en général avec un code non uniforme ce qui ne simplifie pas sa lecture.

Voici quelques règles assez communes :

- Utiliser quatre espaces au lieu des tabulations (ce qui permet d'avoir le même code sur n'importe quelle machine).
- Une ligne ne doit pas faire plus d'un certains nombre de caractères. Le nombre de caractères est en général situé entre 80 et 120. Si une ligne fait plus en général il

vaut mieux la découper sur plusieurs lignes ce qui facilitera sa lecture.

- La façon écrire du code. Par exemple doit on sauter une ligne après la fin d'une méthode, doit on mettre un espace entre une le if et la parenthèse.

Il existe des règles de codage que l'on considère comme des références dans chaque langages qui sont en général décidé avec des membres important de la communauté. Pour illustrer, prenons la norme PSR-2 de PHP<sup>3</sup>. Il existe aussi les conventions pour le langage Java<sup>4</sup>.

Maintenant pour en revenir à l'outil qui gère ces conventions, il a tout sont intérêt car si un développeur ne les respectent pas les erreurs seront tout de suite mises en évidence et il pourra les corriger rapidement se qui évitera que le code ne devienne illisible dans un futur proche, car un code homogène est un réel plus pour sa compréhension et sa maintenabilité.

Quelques exemple d'outils, les outils en général sont nommé CheckStyle dans chaque langage:

Checkstyle pour Java : <https://github.com/checkstyle/checkstyle>

PHPCheckStyle pour PHP: <https://code.google.com/p/phpcheckstyle/>

### 3.9 Serveur d'intégration continue

C'est le centre névralgique d'un système d'intégration continue. C'est lui qui va lancer tous les outils que nous avons vu précédemment et en tirer un certain nombre de tendances. Les possibilités des serveurs d'intégrations continu sont quasi infinie car en général il s'agit de boîte plus ou moins vide dans lequel le développeur est libre de mettre ce qu'il veut afin de coller au mieux à son besoin. Par exemple ça ne pose aucun soucis au serveur d'intégration continu de faire coexister un projet PHP et un projet écrit en Java, vu que chaque projet est indépendant. Chaque projet correspond à

---

<sup>3</sup><https://github.com/php-fig/fig-standards/blob/master/accepted/fr/PSR-2-coding-style-guide.md>

<sup>4</sup><http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-139411.html>

ce que l'on appelle dans le monde de l'intégration continu à un **build**, il s'agit plus ou moins d'un script qui contient tous les outils qui doivent être lancés quand le build est lancé (tests unitaires, code coverage, ...). L'évènement déclencheur est en général le commit du développeur, mais il existe de nombreux autres évènements sur lesquels on peut déclencher un build. Il faut savoir qu'il est aussi possible de programmer un build par un système de cron (ex: lancer ce build le samedi à minuit). Une fois que le build est terminé le serveur d'intégration continu va traiter les données qu'il a tirées du build, pour le présenter à l'utilisateur sous forme en général de graphiques et de tableaux (évolution du taux de couverture, ...). Ce qui va permettre au client et au chef et bien évidemment aux développeurs de voir quels points il va falloir corriger au plus vite. Bien évidemment en cas d'échec du build un mail peut être envoyé aux développeurs afin qu'il règle les soucis les plus graves sur le code du projet. Les critères d'échec d'un build peuvent être multiples et être configurés. Par exemple il est possible de faire échouer un build si le taux de couverture des tests est inférieur à 80%.

J'ai parlé ici des logiciels les plus complexes comme **Jenkins** ou **Bamboo**, mais il existe un courant qui est assez répandu aujourd'hui qui consiste à avoir un serveur d'intégration continu minimaliste sur le cloud, mais beaucoup moins paramétrable, mais bien plus simple à l'usage, nous nous y attarderons dans la dernière partie.

Voici des captures d'écrans de Jenkins

The screenshot shows the Jenkins web interface. At the top left is the Jenkins logo. Below it is a sidebar with links: New Job, Manage Jenkins, People, Build History, Redmine, and My Views. The main area displays 'Continuous Integration Build Server.' and a table of build jobs. The table has columns for S (Success), W (Warning), Job, Last Success, Last Failure, and Last Duration. Three jobs are listed: Project B, Project A, and Project C. Project B has a last success of 1 hr (#26) and a last failure of 2 hr (#24). Project A has a last success of 1 mo 5 days (#31) and a last failure of 3 hr (#25). Project C has a last success of 28 days (#32) and a last failure of 3 mo 0 days (#23). Below the table is a legend for the icons: S for all, W for failures, and L for just latest builds. The footer shows 'Page generated: Jun 22, 2011 9:04:00 AM' and 'Jenkins ver. 1.399'.

**Jenkins**

search ?

ENABLE AUTO REFRESH

Continuous Integration Build Server.

[edit description](#)

**All** +

S	W	Job	Last Success	Last Failure	Last Duration
		<a href="#">Project B</a>	1 hr (#26)	2 hr (#24)	18 sec
		<a href="#">Project A</a>	1 mo 5 days (#31)	3 hr (#25)	4.2 sec
		<a href="#">Project C</a>	28 days (#32)	3 mo 0 days (#23)	3.5 sec

Icon: [S](#) [M](#) [L](#)

Legend for all for failures for just latest builds

**Build Queue**

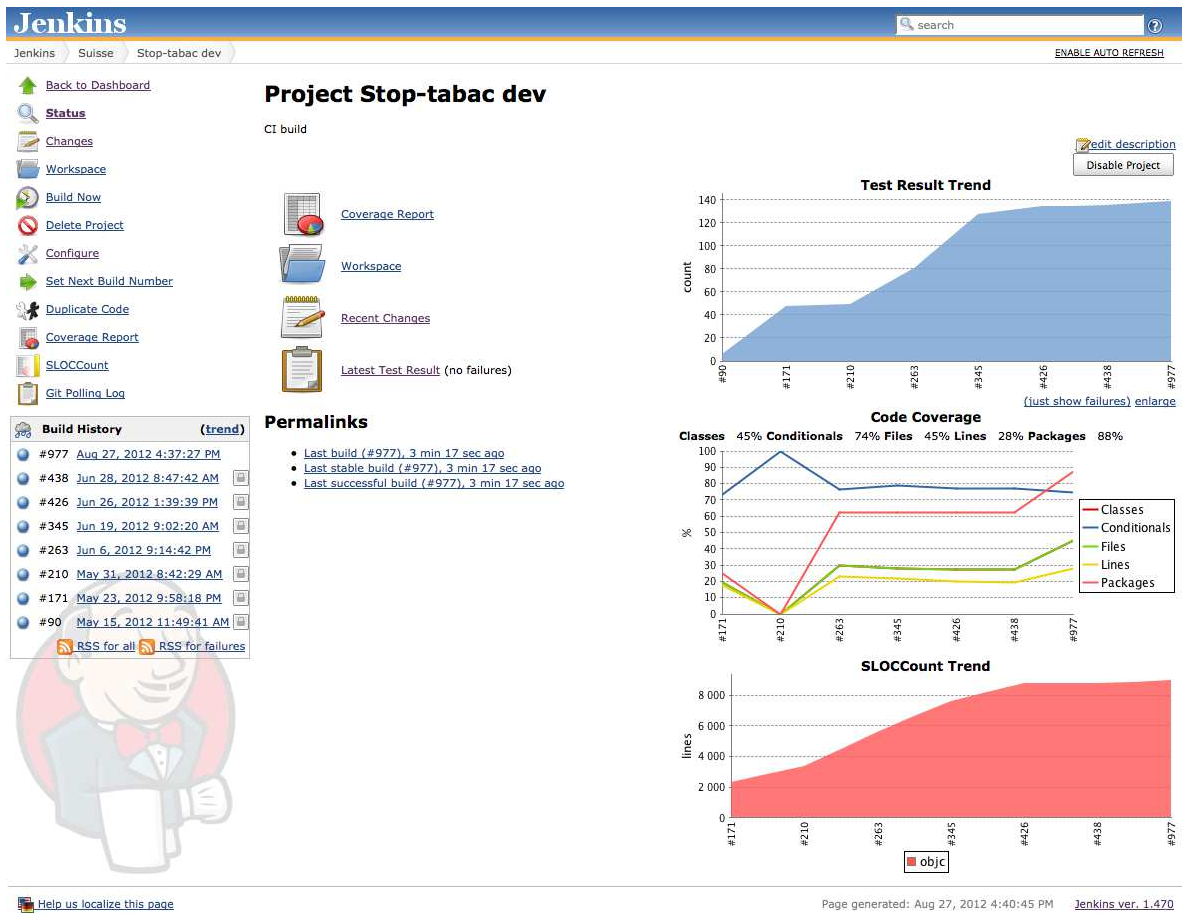
No builds in the queue.

**Build Executor Status**

#	Status
1	Idle
2	Idle

Page generated: Jun 22, 2011 9:04:00 AM [Jenkins ver. 1.399](#)

Dans cette capture nous voyons tous les projets qui sont scanné par Jenkins en ce moment cette page permet d'avoir une vue global des projets et de leurs états (ce projet compile-t-il ou pas?).



Dans cette image nous voyons les statistiques sur le long terme du projet. Ce qui permet de ce faire une idée rapide de la qualité du code, et prendre des mesures efficace afin d'améliorer ces résultat avant qu'il ne soit trop tard pour agir.

### 3.10 Conclusion

Dans cette conclusion, je souhaiterais donner un ordre d'importance à tous les outils car en effet, ils n'ont pas tous le même intérêt ni la même importance. Je vais donc cité ici les outils vraiment indispensables pour avoir une bonne base pour son intégration continue.

Dans un premier temps il faut absolument versionner son code si cela n'est pas fait même pas la peine de penser à l'intégration continue, l'utilisation d'un gestionnaire de sources décentralisé peut être un plus. Utiliser les outils de tests unitaires, avant d'écrire



toutes sortes de tests plus ou moins utile (fonctionnel, GUI, etc...) il faut absolument écrire des tests unitaires qui valideront à chaque modifications que le code est toujours aussi stable qu'avant. Enfin il vous faudra bien évidemment un serveur d'intégration continue qui sera chargé de faire le lien entre les différents outils cité précédemment.

Si les outils que j'ai cités ci-dessus sont les plus importants cela ne signifient pas que les autres ne sont pas utiles à l'intégration continue et qu'ils ne doivent pas être installés. Cela signifie qu'ils ne sont pas nécessaires pour les petites équipes ou les petits projets, ou qu'ils peuvent être facilement intégrés par la suite dans le workflow de son intégration continue si on veut augmenter la qualité du code produit. En effet les outils que j'ai cités comme nécessaire le sont surtout car, sans eux il serait impossible de parler d'intégration continue. Il peut être aussi intéressant de garder uniquement très peu de technologie pour éviter de dérouter les développeurs en les assommant de nouvelles contraintes, puis à mesure que le projet grossit et que les développeurs s'habituent à incorporer de nouveaux outils, en bref rendre cela ludique et leur montrer que l'intégration continue n'est pas juste quelques choses qui sanctionnent, mais aussi un support d'amélioration pour les développeurs. Qu'y a-t-il de plus gratifiant que de voir la courbe de qualité de son projet monté !

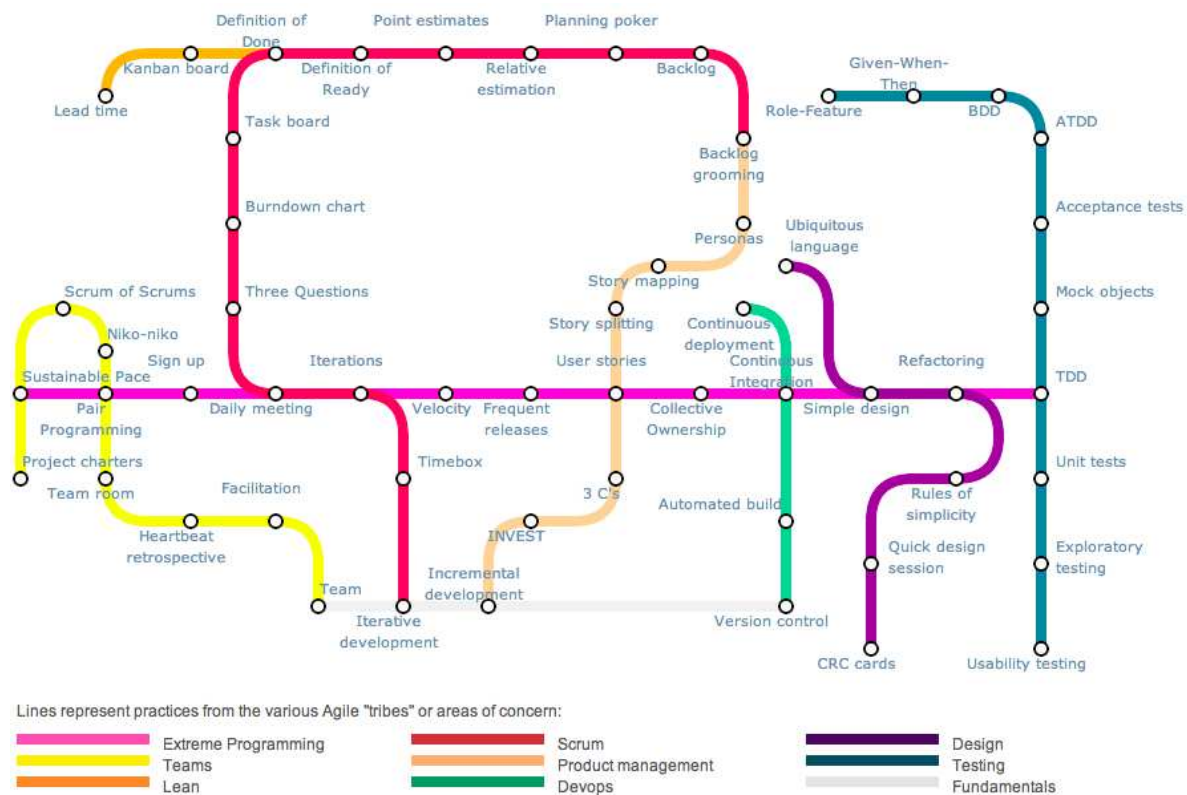
## 4 Les méthodes pour l'intégration continue

En général dès que l'on veut mettre en place tous les outils permettant d'écrire du code de qualité. Les chefs de projet se rendent compte très vite lors de leur premier projet utilisant l'intégration continue que les résultats escomptés ne sont pas là. Pourquoi ?

Bien souvent les équipes qui n'utilisent pas l'intégration continuent n'ont jamais connu d'autre méthodes de développement que l'antique cycle en V. Mais pour qu'un projet utilise tout le potentiel de l'intégration continue il faudra changer les habitudes de l'équipe et utiliser des méthodes de développement bien plus modernes. C'est pour cela que dans cette partie nous verrons les méthodes de développement qui se marie le mieux avec le concept d'intégration continue. Il existe une catégorie de méthode de développement qui correspond exactement à la philosophie de l'intégration continue, il

s'agit des méthodes Agiles, c'est pour cela que toutes les méthodes présentées ci-dessous seront des méthodes agiles.

L'Agile Alliance propose sur son site une liste complète de tous les pratiques agiles divisées en neuf catégories sous forme d'une map<sup>5</sup>. Ces pratiques agiles sont plus ou moins connecter en eux en ayant pour objectif de simplifier les méthodes utilisées actuellement en entreprise afin d'avoir un rendement plus productif. Cette map de l'agilité (voir ci-dessous) nous explique pour chaque méthode agile qu'elles sont les pratiques utilisées.



La méthodologie agile permet de créer un logiciel de façon incrémentielle. Ceci est obtenu en ajoutant de nouvelle issue à chaque itération, mais aussi par la refactorisation du code existant écrit au cours des itérations précédentes. Ce remaniement peut être réalisé en toute sécurité que si vous avez un système de test solide en mesure de vérifier que le produit logiciel ne se cassent pas lorsque vous ajoutez du nouveau code, ou lorsque vous modifier le code existants. Par conséquent, lorsque vous développez votre logiciel,

<sup>5</sup><http://guide.agilealliance.org/subway.html>

vous devez créer deux systèmes distincts, mais fortement liés qui sera la recette de votre application :

- L'application que vous souhaitez vendre à vos utilisateurs
- Son harnais de test qui vous aide à construire de façon incrémentale et agile

Dans un premier temps nous parlerons des méthodes Agiles les plus communes et utilisées comme SCRUM et Extreme Programming qui sont des méthodes compatibles à l'intégration continue. Puis dans les deux parties suivantes deux autres techniques de développement qui peuvent être alliés aux méthodes agiles qui permettent de profiter au maximum de l'intégration continue.

## 4.1 Méthode agiles

Dans cette partie nous allons dans un premier temps expliquer ce qu'est une méthode agile ensuite nous verrons les méthodes agiles les plus utilisées afin de vous convaincre à vous lancer dans la mise en oeuvre de méthode agile.

Les méthodes agiles se veulent en rupture avec la gestion de projets des débuts (cycle en V, cascade, etc.). Le plus gros écueil de ces méthodes était ce que l'on appelle l'effet "tunnel", c'est-à-dire que le client ne voit plus rien des développements une fois qu'il avait validé les besoins et les spécifications de son projet. Ce n'est qu'à la fin qu'il découvre ce qui a été fait, deux cas arrivent fréquemment :

- Soit le projet subit de gros retard car toutes les spécifications initiales étaient incomplètes ou irréalisables et donc n'ont pas été finalisé à temps, dans ce cas le client n'a rien et dépend complètement de l'équipe technique.
- Soit au final les besoins du client ont changé entre la validation des spécifications et la livraison, dans ce cas tout le développement est alors jeté pour être recommencé (ce cas est appelé "Scope creep"<sup>6</sup>)

---

<sup>6</sup>[http://en.wikipedia.org/wiki/Scope\\_creep](http://en.wikipedia.org/wiki/Scope_creep)

C'est donc en se basant sur ce postulat que les méthodes agiles ont été créées, donc le point commun de toutes méthodes agile est de faire disparaître l'effet "tunnel". Pour cela on se base sur plusieurs principes :

- Transparence : Le client verra l'avancement au fur et à mesure de l'avancée du projet.
- Moins de documentation : Écrire du code plus clair plus lisible mais moins bien documenté car écrire de la documentation n'est en général jamais lu (ou très peu).
- Flexibilité : le client doit être très impliqué dans le projet il se doit de faire des retours (positifs et/ou négatifs). En contreparties il peut faire évoluer ses besoins (ajout de fonctionnalité, etc.. en cours de développement.

Actuellement, les 3 méthode agile les plus utilisée dans le cadre de la réalisation de projets informatiques sont le Kanban, Scrum et eXtreme Programming, ce sont ces méthode que nous allons vous expliquer brièvement.

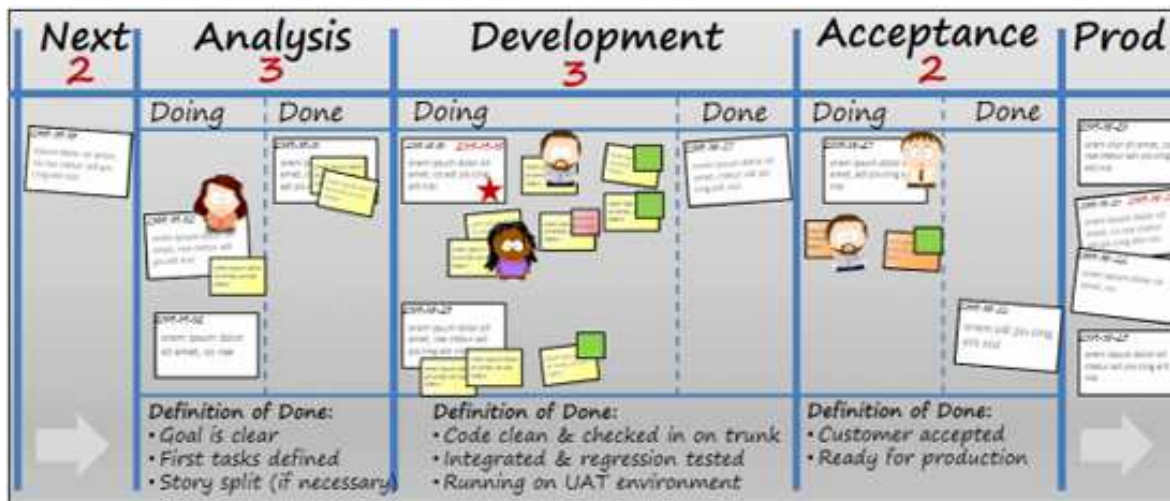
#### 4.1.1 Kanban

Cette méthode, issue de la méthode industrielle Lean<sup>7</sup> est un terme japonais signifiant "fiche" ou "étiquette" mise en place dans les différentes usines de Toyota dans les années 60. Cette méthode permet de visualiser l'état des différentes issues d'un projet visuellement grace a un tableau. Chaque issue possède une mesure nommée "lead-time" qui correspond a la durée moyenne pour completer un item.

Ci dessous un exemple de tableau Kanban :

---

<sup>7</sup><http://fr.wikipedia.org/wiki/Lean>



Cette méthode est généralement utilisée avec la méthode Scrum.

#### 4.1.2 Scrum

Publiée en 2001 par Ken Schwaber et Mike Beedle, la méthode Scrum n'est pas à priori une méthode mais plutôt une méthodologie de gestion de projet agile que les entreprises doivent utiliser afin de surmonter les problèmes et proposer un cadre de gestion de projets Agiles : un rythme itératif (sprint), des réunions précises (daily meeting), des artefacts (product backlog, sprint backlog, graphique d'avancement), des règles du jeu et bien plus encore. Scrum nous donne le processus qui nous mènera de la création d'une vision du produit final, quel que soit le processus de développement réel. La méthodologie Scrum ne nous dit pas comment créer un logiciel de qualité. Il ne montre ce que la qualité est, où sont vos problèmes, et vous met au défi de les corriger.

La méthode Scrum possède un lexique bien particuliers, par exemple dans cette liste non exhaustive :

- Les Rôles
  - Product Owner : Généralement un expert du domaine métier du projet.
  - Equipe de Développement : Ce sont les développeurs, architecte logiciel, graphiste, analyste fonctionnel, ...

- Scrum Master : C'est le coach du Product Owner et de l'équipe de Développement
- Les artefacts
  - Planning poker : Une méthode qui permet de produire des estimation sur les tâches a effectuer
  - Product Backlog : Liste des fonctionnalités à développer
  - Sprint : Une itération d'une durée 2 à 4 semaine pour développer la liste des issues du backlog définie en début du sprint
  - Sprint Backlog : Represente la liste des tâches accomplies pendant le sprint
  - Daily Scrum : Une réunion quotidienne de 15 à 20 minutes maximum
  - Burndown Chart : Represente graphiquement l'avancement du projet

Scrum consiste à diviser les tâches en Sprint, chaque sprint dure généralement 2 semaines cependant elle peuvent être entre quelques heures jusqu'à un mois.

Cette méthode offre seulement des aspects de gestion de projets aidant les développeur de mieux appréhender le développement en surmontant les obstacles, de comment développer et de comment spécifier mais pour combler le problème de la pratique de développement on utilise généralement la méthode Extreme Programming (XP) qui offre des pratiques de programmation en binôme, des développements pilotés par les tests (Test Driven Development), intégration continue, etc.

#### **4.1.3 eXtreme Programming (XP)**

Publiée en 1999 par Kent Beck, Ward Cunningham et Ron Jeffries, la Programmation Extreme est une méthode agile de développement de logiciels méthodologie. Il nous donne un processus qui permet de créer des logiciels de manière agile et productive. Il traite, mais ne se spécialisent pas dans la gestion du processus de développement, et se concentre principalement sur les pratiques d'ingénierie nécessaires pour fournir des logiciels, avec la qualité. XP se compose d'un certain nombre de pratiques méthodologie, conçus pour être utilisés ensemble. Par exemple le TDD, cependant les organisations ne

veulent pas nécessairement toute adopter l'eXtreme Programming et préfère généralement la méthode Scrum combiné avec le Kaban.

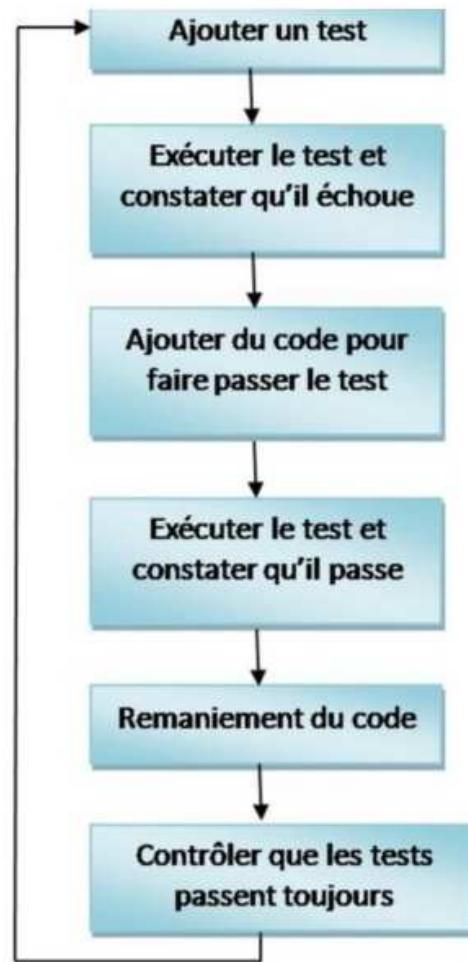
## 4.2 Le développement piloté par les tests (TDD)

Quand on est en phase de développement, on n'a pas toujours le temps de créer les tests adéquats pour son bon fonctionnement et on oublie souvent de les faire faute de temps. Sur certain projet on prend des gens qui ne coûtent pas cher et qui ne sont pas développeurs afin de tester le cas fonctionnel des nouvelles fonctionnalités, dans certains projets c'est le chef de projet ou le scrum master qui teste la fonctionnalité rapidement. Mais ceci est une mauvaise pratique car d'ici quelque semaine voir quelque mois, quand une fonctionnalité importante ne fonctionne plus, on se demandera depuis combien de temps elle est comme cela et surtout pourquoi!

En utilisant des tests de régression, le problème serait résolu, mais il faut cependant les écrire et cela représente un budget supplémentaire lors de l'écriture de ces tests, une autre solution existe qui consiste à utiliser la méthode TDD ("Test Driven Development" ou en français "Le développement piloté par les tests") permet d'écrire le test avant le code. Le développeur conçoit un ensemble de tests pour la première "user Story" du cahier des charges, certes ces tests échouent mais c'est le but car on devra faire en sorte qu'il fonctionne en écrivant le minimum de code possible. Généralement un développeur aura tendance à écrire plus de code que le nécessaire cependant avec le processus de TDD, le développeur écrira le minimum de code et si les tests passent alors ce code sont considérés comme solides et conforme au cahier des charges.

Ainsi le TDD permet de concevoir du code en état de marche à n'importe quel moment du développement du projet en plus d'améliorer la qualité du code et nous rassure que le besoin du client est respecté.

Pour résumer ce qui vient d'être dit voici un schéma qui explique les étapes à suivre en TDD:



## 4.3 Behavior Driven Development (BDD)

### 4.3.1 C'est quoi?

Il faut préciser dès à présent que cette méthode de développement se base en grande partie sur les principes de la TDD vu précédemment, donc ici aussi il faudra créer des tests unitaire avant de coder, ici c'est la personne et la manière de créer ces tests qui diffère, en effet ici aussi il sera question de User Story et de Features. Le Behavior Driven développement est née pour répondre à une question simple: **qui connaît le mieux ce qui doit être développé?**. La réponse est bien évidemment le client, mais bien souvent le client n'est pas un développeur, et il n'est pas capable d'écrire les tests

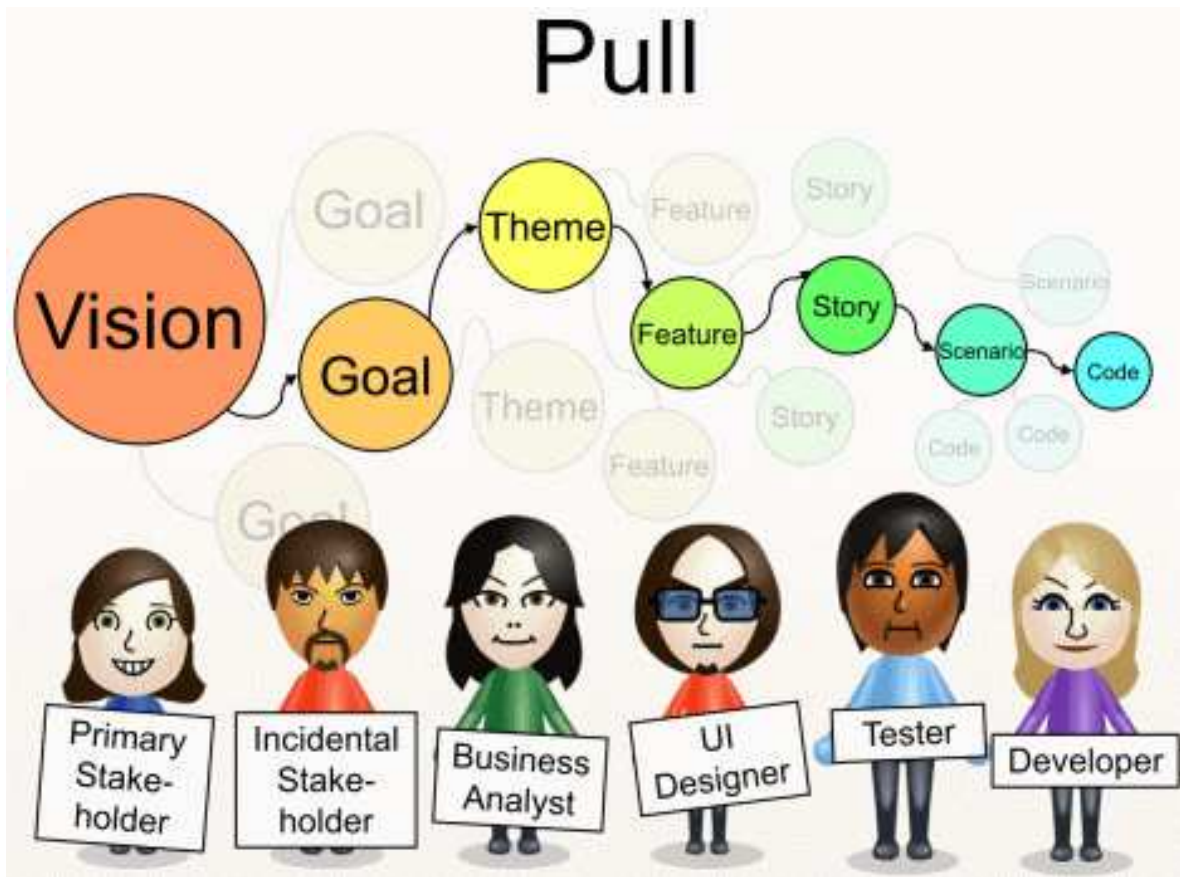


unitaire nécessaire pour tester le comportement de son application. Lorsqu'on utilise le BDD c'est bien le client qui va créer les tests unitaires, bien évidemment ce n'est pas directement lui qui va coder, mais un logiciel qui va traduire ce qu'il veut en code. Nous vous présenterons deux de ces logiciels plus tard. Bien évidemment cela oblige le client à s'investir réellement dans son projet afin qu'il soit pleinement satisfait, car sinon dans le cas contraire le BDD ne fonctionnera pas en effet comme c'est au client de décrire très précisément ce qu'il souhaite si cette tâche n'est pas effectuée avec un minimum de rigueur les fonctionnalités attendues pour le client pourraient être erronées.

#### **4.3.2 Comment ça marche?**

Et bien rien de très compliqué, le client écrit sous forme de texte les fonctionnalités (Features) qu'il attend sous la forme d'un langage compréhensible par le logiciel de BDD, puis pour chaque fonctionnalité il écrit un certain nombre de scénarios visant à tester le comportement qu'il souhaite donner à cette Feature. Une fois un certain nombre de scénarios écrits pour la feature le développeur code la-dite feature puis grâce à son logiciel de BDD celui-ci va générer des tests unitaires qui correspondent à aux scénarios (cette partie est invisible aux yeux du développeur), afin de tester si le code de la feature a bien été implémenté.

Voici un schéma qui explique cela de manière imagée:



### 4.3.3 Les solutions existante

Maintenant nous allons voir deux solutions sur lesquelles on peut passer le Behavior Driven Développement. Ici nous n'en présentent que deux mais il en existe bien d'autres. Cette méthode de développement étant assez récente il est possible qu'il soit assez difficile de trouver beaucoup de ressources fiables pour votre langage favori.

**Cucumber** C'est la solution la plus ancienne et la plus utilisée, elle est disponible dans quasi tous les langages de programmation, du Java en passant par PHP ou même Python et Ruby. Si vous n'êtes pas très au fait des technologies BDD qui existe dans votre langage de programmation c'est celui-ci qu'il faut choisir.

**Behat** C'est une solution exclusivement php qui a son propre langage d'écriture de test en PHP (inspiré de Cucumber), ce logiciel à une très forte communauté derrière lui. Il est à conseiller à tous les développeurs PHP, car il s'agit d'une solution dédiée au PHP contrairement à Cucumber.

#### 4.3.4 Exemple complet

Dans cet exemple nous allons voir à quoi ressemble le fichier de description pour Behat qui s'inspire très fortement de la syntaxe utilisé par Cucumber. On remarquera d'ailleurs qu'on utilise le même vocabulaire que dans la TDD (Features, Scenario, ...)

Feature: ls

In order to see the directory structure

As a UNIX user

I need to be able to list the current directory's contents

Scenario:

Given I am in a directory "test"

And I have a file named "foo"

And I have a file named "bar"

When I run "ls"

Then I should get:

"""

bar

foo

"""

Dans l'exemple ci-dessus on crée le un scenario pour la commande `ls` (qui permet de lister le contenu d'un dossier). Nous voyons qu'il est simple de comprendre ce que l'on attend à la fin de cette commande pour le scénario donné. De plus l'écriture de ce scénario est bien plus rapide que l'écriture d'une classe de test unitaire.

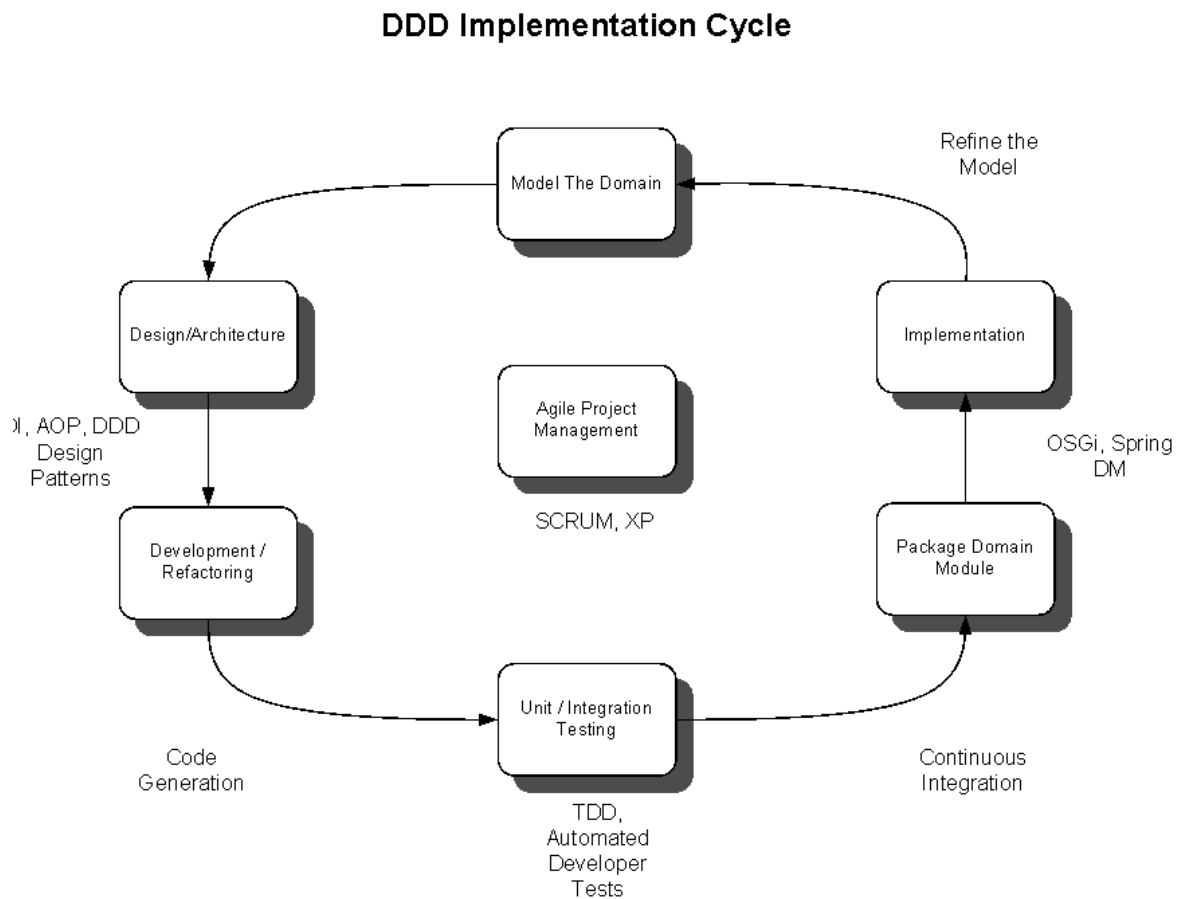
## 4.4 Conception pilotée par le domaine (DDD)

Ici nous allons voir une méthode de qui n'est pas encore très répandue dans le monde de l'agile, il s'agit du DDD (Domain Driven Development), il s'agit d'une méthode de développement Agile qui vise à réduire le plus possible les erreurs de conceptions des logiciels, nous ne nous attarderons pas beaucoup sur cette partie car elle est encore trop jeune.

### 4.4.1 Le principe du DDD

Lorsque que l'on fait du DDD on part du principe que le plus important c'est de bien connaître quoi et pourquoi on développe, le principe premier est donc comme dans le BDD de bien connaître son domaine de gestion, mais ici le but n'est plus de le tester du code qui va être développé, mais directement la génération d'un code qui va faire ce qu'on va lui demander, et où le développeur n'aura plus qu'à finir le code qui a précédemment été généré.

Voici le cycle de vie d'un projet qui utilise le DDD:



## 5 Les nouveautés

### 5.1 Ça va bientôt arriver dans le cloud !

Vous n'avez pas pu y lopper depuis quelques années déjà la mode et au cloud, tout se trouve sur Internet (messagerie, album photo, documents). Pourquoi cette mode d'utiliser le cloud ? La réponse est simple, c'est pratique et souvent moins onéreux. On avait déjà remarquer une démocratisation du développement sur le cloud avec GitHub<sup>8</sup> le fameux site qui utilise le gestionnaire de source décentralisé GIT dont nous avons parlé dans l'une des parties précédentes.

Maintenant c'est au domaine de l'intégration continue de s'y mettre. Bien souvent couplé

<sup>8</sup><http://github.com>

à GitHub. Car, ce genre de pratique est né avec la démocratisation du développement Open-Source sur GitHub. D'ailleurs, GitHub a réussi à rendre plus simple la mise en commun et la contribution du code sources pour des projets Open-Source.

Nous allons donc parler ici de deux grosses pratiques qui sont déjà bien ancrée dans le monde de l'open-source mais qui commence à bien s'implanter chez les entreprises. Dans un premier temps nous vous parlerons de la livraison continue une pratique encore peu connue mais très novatrice qui profite fortement des possibilités du cloud. Puis, dans un second et dernier temps nous verrons les serveurs d'intégrations continuent sur le cloud.

### **5.1.1 La livraison continue**

La livraison continue ("Continuous Delivery") est une pratique qui propose de développer une application afin qu'elle puisse être livrer dans l'environnement de production d'un client.

Ce principe repose sur deux grand fondement :

- Automatisation des tâches, permet d'avoir une procédure de déploiement fiable sans problème lier souvent aux différents oublie des développeurs.
- Déployer fréquemment, permet de recevoir des feedback sur l'intégration de l'application sur le serveur ainsi les utilisateurs peuvent bénéficier rapidement des nouveautés de même les développeurs peuvent revoir et corriger leur code source quand il est toujours chaud en mémoire.

En commençant par un correctif ou l'ajout d'une nouvelle fonctionnalité apportés au code jusqu'à son déploiement en production est appelé le « pipeline de déploiement ». Ainsi, en réduisant le délais entre la demande du client et la mise à disposition de la nouvelle fonctionnalité ou d'un correctif d'un bug, la satisfaction du client augmente.

### **5.1.2 Intégration continue**

L'intégration continue ne correspond pas à une seule pratique mais à un ensemble de pratiques et d'outils qui permet de vérifier le bon fonctionnement du code à chaque

commit du développeur ainsi cela réduit les risques par exemple de régression. Ces outils que l'on devait installer sur des serveurs dédiés de l'entreprise étaient fastueux à configurer surtout pour les nouvelles entreprises qui se mettent à cette pratique, mais cette époque est révolue car depuis quelques années des services dans le cloud permettent d'éviter d'installer tous ces outils. Le principale avantage étant pour les entreprises novices dans ce domaine d'utiliser l'intégration continue et l'adopter simplement, rien n'empêche une équipe de se défaire de ces outils dans le cloud et d'utiliser des outils d'intégration ne continue sur leurs propres serveurs.

Dans le cloud il existe une multitude d'outils dont voici les plus connu et permettent tous une integration avec Github :

- Google AppEngine<sup>9</sup> : Hebergeur d'application Python, Java, Groovy, JRuby, Scala, Clojure, Go, et PHP
- Heroku<sup>10</sup> : Herberge du Ruby, Node.js, Python, and Java
- Travis-CI<sup>11</sup> : Un jenkins gratuit pour les projets open source

Ci-dessous quelques image de Travis-CI qui est une des valeurs sûr dans ce domaine

**mhor/MhorCvToPdfBundle** build passing

Easy way for generate beautiful CV

Current | Build History | Pull Requests | Branch Summary

Build	48	Commit	3aa921e (master)
State	Passed	Compare	b203e4a527d9...3aa921e01d26
Finished	about 4 hours ago	Author	Maxime Horcholle
Duration	3 min 46 sec	Committer	Maxime Horcholle
Message	remove exit remplace it by return		

**Build Matrix**



Job	Duration	Finished	PHP
48.1	1 min 13 sec	about 4 hours ago	5.3.3
48.2	1 min 26 sec	about 4 hours ago	5.3
48.3	1 min 7 sec	about 4 hours ago	5.4

<sup>9</sup>Google AppEngine : <https://developers.google.com/appengine/>

<sup>10</sup>Heroku : <https://www.heroku.com/>










<sup>11</sup>Travis-CI : <https://travis-ci.org/>

Voici la page principal d'un projet, elle montre le dernier build du projet, on voit que cette interface est beaucoup plus dépouillé que celle de Jenkins, ici il faut oublié tout le coté statistique.

**mhor/MhorCvToPdfBundle**  build passing 

Easy way for generate beautiful CV

Current Build History Pull Requests Branch Summary

Build	Message	Commit	Committer	PR	Duration	Finished
 44	add qrcode generator	<a href="#">ae3a3c3 (master)</a>	mhor	<a href="#">#7</a>	3 min 9 sec	20 days ago
 39	little modifs	<a href="#">6d6636a (master)</a>	mhor	<a href="#">#6</a>	2 min 59 sec	26 days ago
 34	Code Quality	<a href="#">16ef90e (master)</a>	mhor	<a href="#">#5</a>	7 min 29 sec	about a month ago
 31	finally re-add Configuration.php	<a href="#">1cdbcac (master)</a>	mhor	<a href="#">#4</a>	4 min 59 sec	about a month ago
 29	Code Quality	<a href="#">b912c22 (master)</a>	mhor	<a href="#">#4</a>	7 min 19 sec	about a month ago
 24	add mock AppKernel	<a href="#">529c96b (master)</a>	mhor	<a href="#">#3</a>	5 min 50 sec	about a month ago
 19	fix unit test	<a href="#">d7be4f1 (master)</a>	mhor	<a href="#">#2</a>	5 min 22 sec	about a month ago
 17	add unit test for commnd	<a href="#">5c6b49b (master)</a>	mhor	<a href="#">#2</a>	5 min 22 sec	about a month ago
 14	remove until comment	<a href="#">e6dd3f1 (master)</a>	mhor	<a href="#">#1</a>	4 min 59 sec	about a month ago

Show more

Lorsque l'on utilise Git, il est très pratique après avoir développer sa feature d'effectuer un pull request et ainsi vérifier si le code compile toujours avant d'appliquer les modifications dans la branche principale.

## 5.2 L'Intégration continue en local

Depuis quelque temps, nous avons la possibilité d'installer les outils en local concernant l'intégration continue. Cette pratique en local permet de voir l'avancement ainsi que les différents test de l'application (test d'intégration, test unitaire, ...) avant de commiter ainsi le developpeur pourra voir si son code fonctionne correctement, que les bonnes pratiques soit bien pratiquer et que son code soit complètement tester grace a un outils de coverage. Une application couplé avec un dépôt Git, offre la possibilité de crée des Git Hooks<sup>12</sup>

<sup>12</sup>Git Hooks : <http://git-scm.com/book/ch7-3.html>



Les outils d'intégration continue en local se multiplie ainsi on peut nommée comme outils Emma<sup>13</sup> pour une gestion du coverage du code pour des applications java et son equivalent pour des applications PHP, sismo<sup>14</sup>

EMMA Coverage Report (generated Sun Jan 12 20:29:15 CET 2014)				
[all classes]				
COVERAGE SUMMARY FOR PACKAGE [fr.esglia]				
name	class, %	method, %	block, %	line, %
fr.esglia	100% (14/14)	57% (73/129)	47% (1644/3495)	46% (408.5/891)
COVERAGE BREAKDOWN BY SOURCE FILE				
name	class, %	method, %	block, %	line, %
AlphaBeta.java	100% (1/1)	40% (2/5)	3% (8/240)	8% (3/40)
Reine.java	100% (1/1)	67% (2/3)	18% (28/157)	16% (7/44)
Chevalier.java	100% (1/1)	67% (2/3)	18% (28/154)	17% (7/41)
Roi.java	100% (1/1)	67% (2/3)	19% (28/147)	17% (7/41)
IA.java	100% (1/1)	83% (5/6)	19% (17/89)	27% (8/30)
Pion.java	100% (1/1)	67% (2/3)	22% (28/129)	24% (7/29)
Fou.java	100% (1/1)	67% (2/3)	23% (28/121)	22% (7/32)
Move.java	100% (1/1)	30% (8/27)	33% (71/213)	41% (26/63)
Piece.java	100% (1/1)	73% (16/22)	41% (137/335)	44% (45/102)
ChessboardValue.java	100% (1/1)	25% (4/16)	42% (134/320)	36% (30.4/84)
Algorithme.java	100% (1/1)	62% (5/8)	68% (19/28)	64% (9/14)
Helper.java	100% (1/1)	57% (4/7)	70% (468/672)	62% (115.7/189)
Chessboard.java	100% (1/1)	80% (16/20)	71% (555/777)	73% (110.7/151)
Tour.java	100% (1/1)	100% (3/3)	84% (95/113)	80% (25.7/32)
[all classes]				
EMMA 2.1.5320 (stable) (C) Vladimir Roubtsov				
60	@Override			
61	public ArrayList<Move> generateMovesForThisPiece(Chessboard chessboard) {			
62				
63	int toX = -1, toY = -1;			
64	ArrayList<Move> moves = new ArrayList<Move>();			
65				
66	String positionPiece = chessboard.getPositionPiece(this);			
67	int getX = Helper.getXFromString(positionPiece);			
68	int getY = Helper.getYFromString(positionPiece);			
69				
70	// 4 direction			
71	for (int direction = 0; direction < 4; direction++)			
72	// Max 8 moves			
73	for (int length = 1; length < 9; length++) {			
74				
75	if (direction == 0) {			
76	toX = getX + length;			
77	toY = getY;			
78	}			
79	if (direction == 1) {			
80	toX = getX;			
81	toY = getY + length;			
82	}			
83	if (direction == 2) {			
84	toX = getX - length;			
85	toY = getY;			
86	}			
87	if (direction == 3) {			
88	toX = getX;			
89	toY = getY - length;			
90	}			
91				
92	Move move = checkThis(toX, toY, chessboard);			
93				
94	// Si d'placement est nul, plus possible de ce d'placer dans			
95	// cette direction pour la tour			
96	if (move != null) {			
97	moves.add(move);			
98	if (move.isAttack())			
99	break;			
100				
101	} else break;			
102				
103	}			
104	return moves;			
105	}			

## 6 Conclusion

Aujourd'hui l'intégration continue est un grand atout pour un projet

Cette pratique au sein de certaines équipes est utilisée avec un moniteur visuel, par exemple un écran dédié à l'affichage du dernier build sur le serveur ainsi que diverses informations. Dans certaines équipes, lorsqu'un développeur casse le build, il reçoit un

<sup>13</sup>Emma : <http://mojo.codehaus.org/emma-maven-plugin/index.html>

<sup>14</sup>Sismo : <http://sismo.sensiolabs.org/>

gage par exemple, rapporter des croissances le lendemain. Cela est un bon moyen pour renforcer les liens au sein d'une équipe.

## **7 Les sources**

### **7.1 Bug Trackers**

Wikipédia : [fr.wikipedia.org/wiki/Logiciel\\_de\\_suivi\\_de\\_problèmes](http://fr.wikipedia.org/wiki/Logiciel_de_suivi_de_problèmes)

Redmine : [www.redmine.org](http://www.redmine.org)

Jira : [www.atlassian.com/fr/software/jira](http://www.atlassian.com/fr/software/jira)

Trello : [www.trello.com](http://www.trello.com)

Asana : [www.asana.com](http://www.asana.com)

### **7.2 Analyseur de code**

<http://www.journaldunet.com/developpeur/expert/49745/les-tests-des-gens-d-en-haut-et-des-gens-d-en-bas.shtml>

### **7.3 Tests de GUI**

[http://en.wikipedia.org/wiki/List\\_of\\_GUI\\_testing\\_tools](http://en.wikipedia.org/wiki/List_of_GUI_testing_tools)

<http://blog.dreamcss.com/tools/gui-testing-tools/>

<http://atatorus.developpez.com/tutoriels/java/test-application-web-avec-selenium/>

### **7.4 Tests d'intégration**

[http://fr.wikipedia.org/wiki/Test\\_d'int%C3%A9gration](http://fr.wikipedia.org/wiki/Test_d'int%C3%A9gration)

## 7.5 Tests fonctionnels

<http://blog.octo.com/demarches-de-tests-fonctionnels/>

## 7.6 BDD

[http://en.wikipedia.org/wiki/Behavior-driven\\_development](http://en.wikipedia.org/wiki/Behavior-driven_development)

<http://behat.org/>

<http://docs.behat.org/>

<http://cukes.info/>

## 7.7 DDD

[http://en.wikipedia.org/wiki/Domain-driven\\_design](http://en.wikipedia.org/wiki/Domain-driven_design)

<http://dddcommunity.org/>

<http://www.methodsandtools.com/archive/archive.php?id=97>

## 7.8 Méthode agile

<http://www.cienum.fr/sites-internet-mobiles/projets/methodologie-de-projets/kanban>

<http://www.crisp.se/file-uploads/kanban-example.pdf>

<http://stackoverflow.com/>

## ANNEXE 2 - LA CHARTE DE L'ETUDIANT : LE PLAGIAT – A REMETTRE AVEC LA VERSION FINALE IMPRIMEE DU MEMOIRE

La contrefaçon est l'appellation juridique du plagiat, sa version condamnable. A ce titre, elle constitue un délit. L'article 335-3 du Code de la propriété intellectuelle en précise la nature : il s'agit de " toute reproduction, représentation ou diffusion, par quelque moyen que ce soit, d'une œuvre de l'esprit en violation des droits d'auteur, tels qu'ils sont définis et réglementés par la loi". Elle est susceptible de donner lieu à des sanctions civiles et pénales.

Ainsi, le plagiat consiste à copier, contrefaire ou falsifier un document sujet à une évaluation et d'utiliser en tout ou partie, l'œuvre d'autrui ou des passages tirés de celle-ci, sans les identifier expressément comme citations et dans l'intention de les faire passer pour siens.

De même, lorsque vous reprenez « mot pour mot » un passage d'un auteur, il faut impérativement le signaler avec des guillemets et indiquer en bas de pages, la source ainsi que son numéro de page.

### Sanctions disciplinaires

Le plagiat est sanctionné par :

- ◆ un 0/20 sur le dossier ou le mémoire de recherche appliquée
- ◆ le passage devant le Conseil de Discipline
- ◆ les sanctions peuvent aller jusqu'à l'exclusion définitive des examens
- ◆ toute récidive peut se traduire par une exclusion temporaire ou définitive de l'établissement.

### Déclaration sur l'honneur - ANNEE SCOLAIRE 2013-2014

A remettre complétée et signée en annexe du mémoire de recherche appliquée

Nom	Prénom	Formation

Nom du Maître de Mémoire.....

Je (nous) soussigné(s) M ..... atteste (ons) avoir pris connaissance du règlement intérieur de l'école et certifie (ons) que le dossier ou mémoire de recherche appliquée ci-joint ne fait l'objet d'aucun plagiat. Par ailleurs, je (nous) m'engage (ons) à respecter les règles du dit règlement intérieur et les sanctions disciplinaires qui en découlent.

Fait à Paris, le  
Signatures de tous les participants au dossier ou mémoire

Précédées de la mention « lu et approuvé »