[DEPRECATED] Guide: Development Tools [TOTRANSLATE]

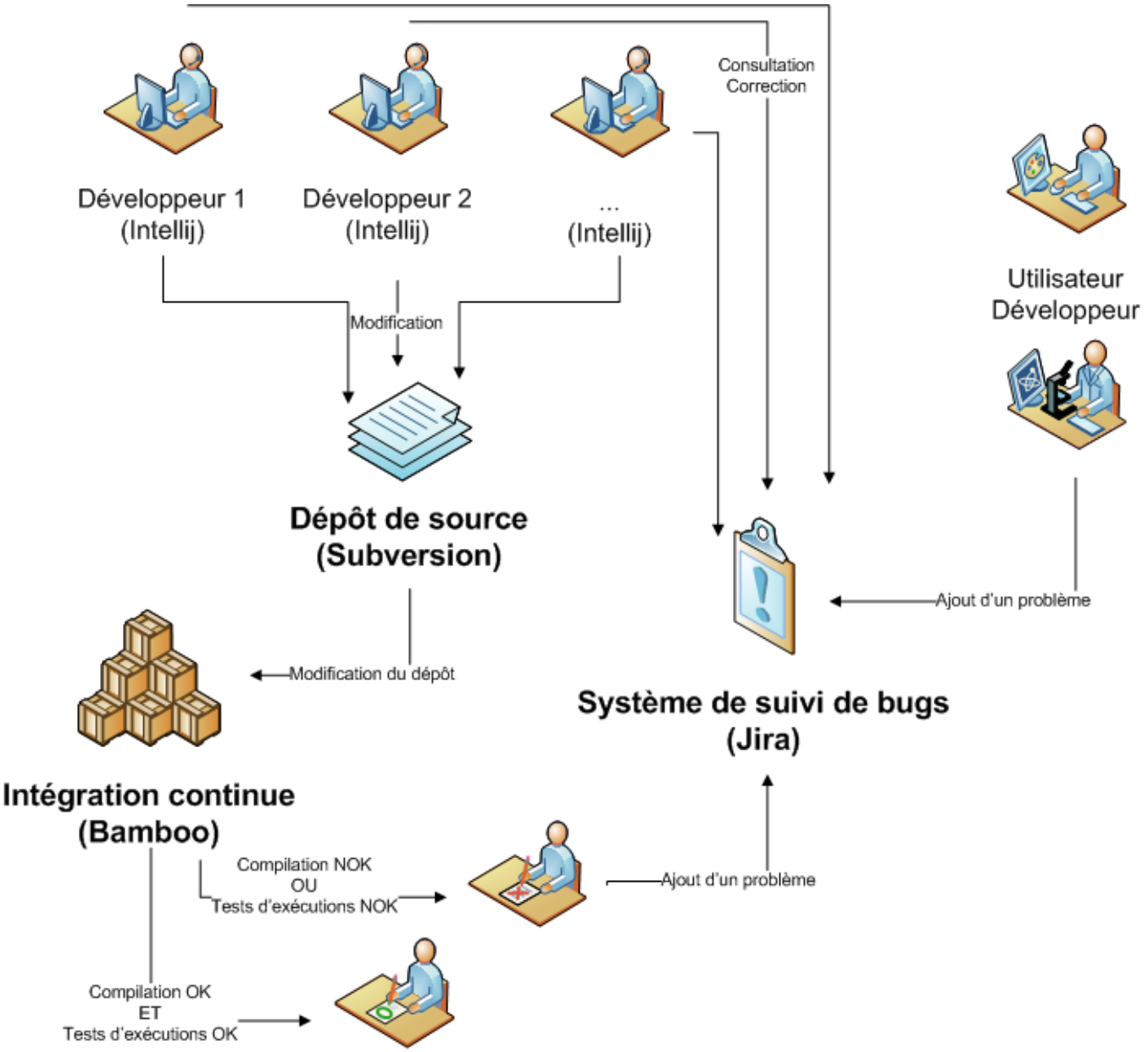
Contexte de développement

Une présentation de nos outils est disponible à cette adresse: La qualité logicielle et l'intégration continue - Cas concret du projet Cytomine

Afin de développer l'application Cytomine facilement à plusieurs développeurs, nous utilisons plusieurs outils:

- Git: dépôt de sources,
- Jira: système de suivi de bugs,
- Bamboo: système d'intégration continue,
- Confluence: wiki,
- Sonar: qualité logicielle

Ci-dessous, une illustration du fonctionnement des outils et une description complète de chacun d'eux.
Rem: subversion a été remplacé par git et Sonar n'apparaît pas.



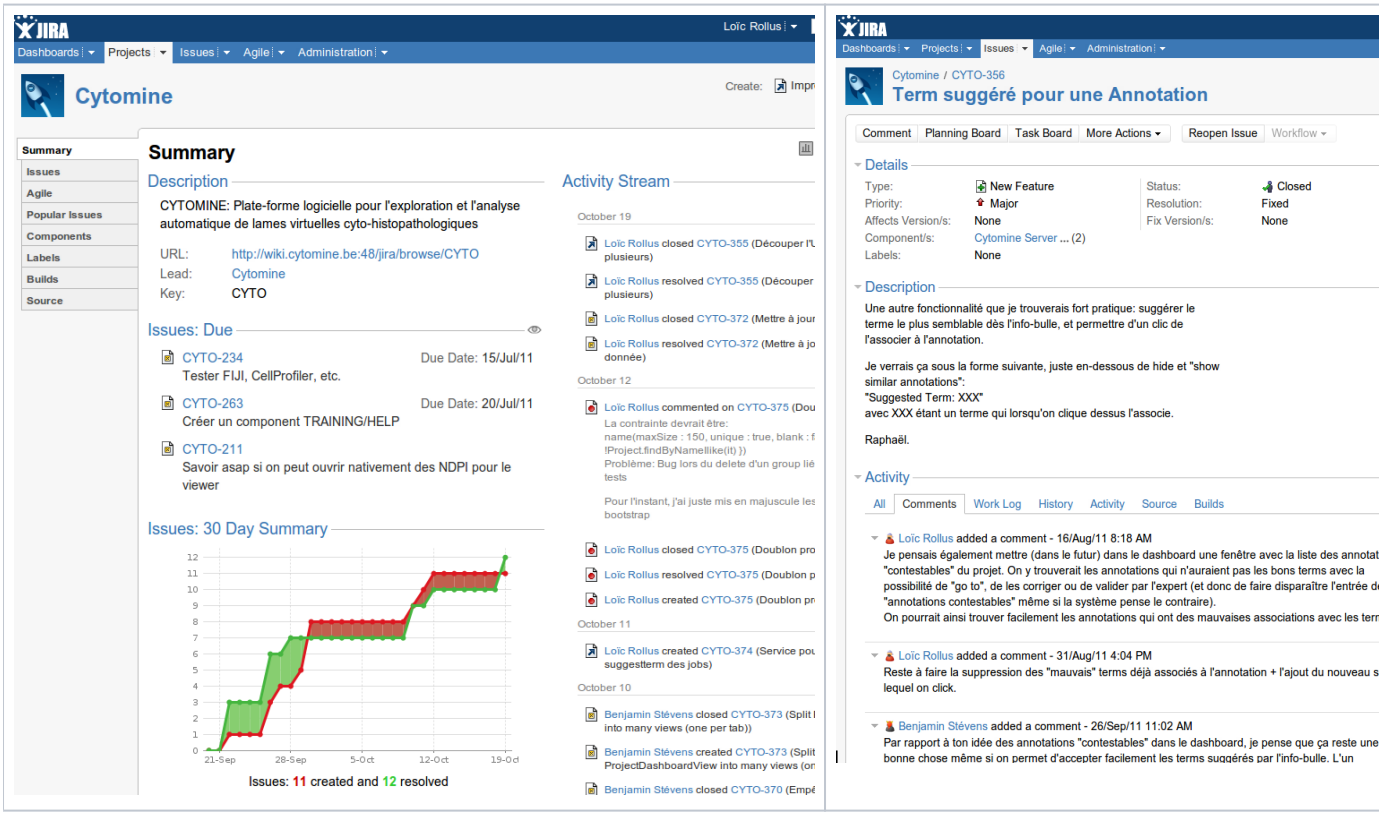
Gestion des sources (Git/Github)

Système de suivi de bugs (Jira)

Un système de suivi de bugs permet aux développeurs, aux testeurs et aux utilisateurs d'améliorer la qualité du logiciel.
Jira fonctionne sous forme de ticket permettant à un utilisateur quelconque (développeur, testeur, utilisateur final,...) de décrire un problème rencontré, une amélioration ou une fonctionnalité souhaitée.
Cette base de données de ticket doit alors être consultée par les développeurs. Ils pourront également prévoir et se répartir facilement les tâches grâce à ce type de système.

Voici le cheminement classique lors d'un bug:

1. L'utilisateur (utilisateur final, développeur ou testeur) ouvre un ticket en signalant le problème et le composant touché.
2. Un développeur prend en charge le ticket.
3. Ce développeur signale qu'il commence a travailler sur ce ticket et corrige les erreurs dans le code.
4. Il propage ses modifications sur le dépôt et signale le ticket comme "résolu".
5. Si la correction est effectué, l'utilisateur ferme le ticket, sinon il peut le réactiver.



Intégration continue (Bamboo)

L'intégration continue consiste à vérifier le code source du logiciel chaque fois qu'il a été modifié par un développeur.
Cette vérification est divisée en plusieurs étapes:

1. Compilation du code source,
2. Lancement de l'application,
3. Exécution des tests d'intégration.

En cas d'échec d'une de ces étapes, la vérification signale que le logiciel n'est pas correct.
Les développeurs, et en particulier le développeur ayant effectué la modification provoquant l'erreur, seront directement averti.

Pour que cette technique soit fonctionnelle, il faut:

- avoir un code source centralisé où les développeurs appliquent leurs modifications fréquemment (ex. via subversion),
- avoir une batterie de tests couvrant le plus possible de code et de fonctionnalités.

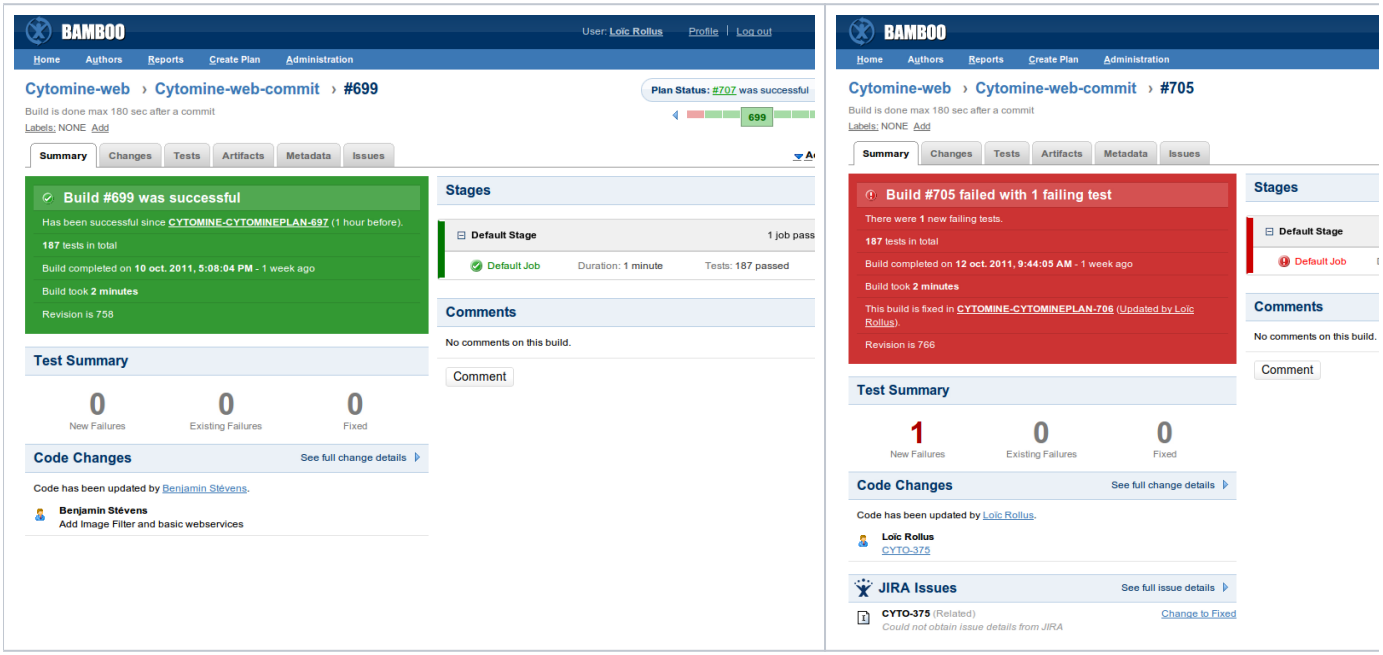
Le processus pour le développeur est le suivant:

1. Ecriture des tests couvrant la fonctionnalité ou le bug détecté.
2. Ajout de la fonctionnalité ou correction du bug.
3. Exécution de la batterie de tests sur sa machine de développement.

4. Si les tests sont effectués avec succès, le développeur peut propager ses modifications sur le dépôts. Sinon, il peut corriger le code et revenir au point précédent.

Les avantages de l'intégration continue couplée à une bonne batterie de tests sont très nombreux:

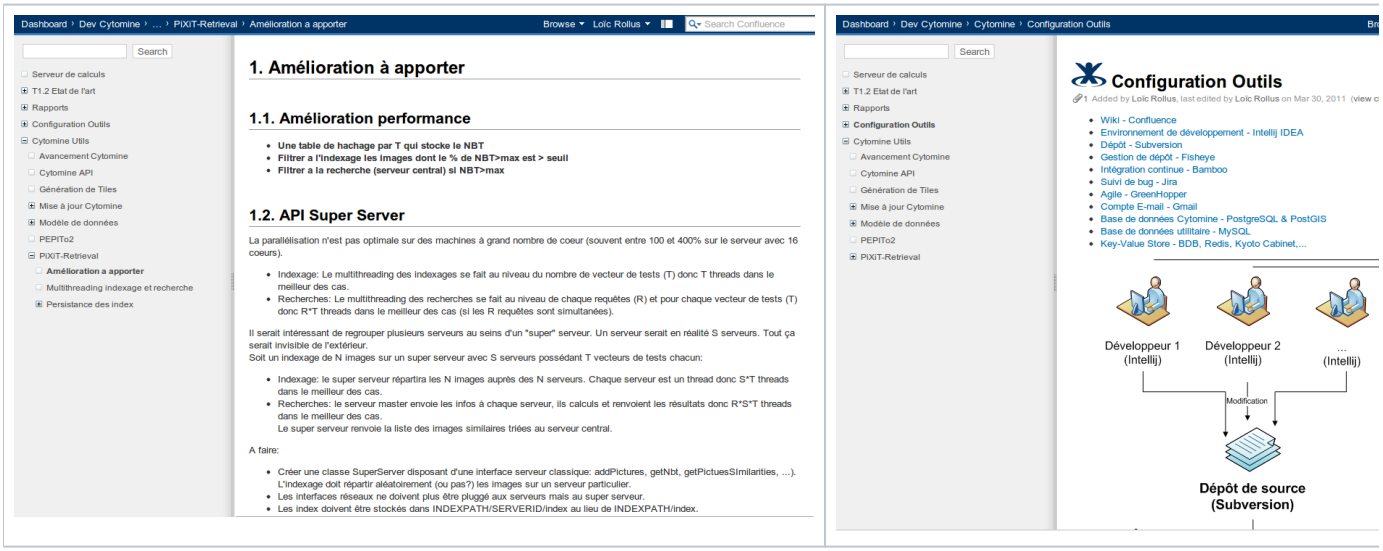
- Les développeurs sont incités à avoir constamment des versions de leurs sources fonctionnelles à 100% dans le dépôt. Si c'est le cas, une version de démonstration incluant les dernières fonctionnalités est toujours disponible.
- Les erreurs et bugs sont détectés juste après l'envoi des modifcations et peuvent être corrigés directement. On évite au développeur de devoir corriger des bugs introduits plus tôt dans le cycle de développement.
- Le code de test peut être écrit avant l'ajout de la fonctionnalité (principe du test driven development ou développement piloté par les tests). Le test permettra alors au développeur de valider ou corriger son code.
- Après le déploiement de ce système, tout est entièrement automatisé et ne requiert presque plus d'intervention humaine.



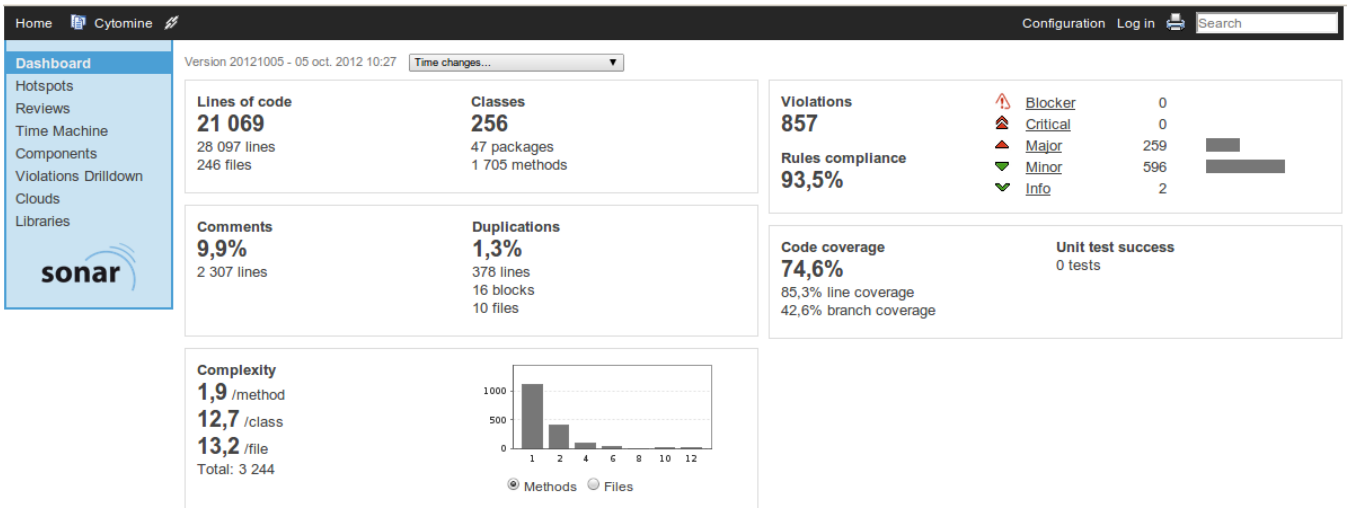
Wiki (Confluence)

Un wiki est un site web dont les pages sont accessibles et modifiables par les utilisateurs. Dans notre environnement, il est utilisé pour plusieurs tâches:

- Echange d'information entre les partenaires (description des besoins, remarques,...),
- Présentation et information sur les outils pour le développement (wiki, intégration continue,...),
- Partage d'information sur le développement de Cytomine (API, Benshmark, ...),
- Rédaction de rapport,
- ...



Qualité logicielle (Sonar)



Description

Outre des statistiques quantitatives (nombre de lignes, classes,...), Sonar permet d'avoir accès a pas mal d'information concernant la qualité du code:

- lignes/blocs de code dupliqués
- un report complet des "problèmes" détectés dans notre code
- la complexité des méthodes
- les rapports sur la réussite des tests
- les rapports sur la couverture des tests
- ...

Un historique complet de l'évolution du code est disponible.

Installation coté serveur

- Téléchargement du serveur Sonar (http://www.sonarsource.org/downloads/)
- sonar.properties: commenter les lignes de la DB par défaut (DB2)
- sonar.properties: décommenter les lignes de la DB Mysql
- exécuter le script de création de DB dans sonar-xxx/extras/databases/

Si installation standalone:

- Démarrage du serveur: bin/OS/sonar.sh start
  Si installation avec tomcat:
- cd sonar-xxx/war
- ant
- coper le war dans tomcat

Installation coté client (machine développeur)

- Créer un fichier pom.xml dans le répertoire root du serveur grails (le répertoire parent de grails-app)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yourcompany</groupId>
  <artifactId>yourproject</artifactId>
  <version>20121005</version>
  <packaging>pom</packaging>
  <name>Cytomine</name>
  <build>
    <sourceDirectory>grails-app</sourceDirectory>
        <plugins>
           <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-compiler-plugin</artifactId>
              <configuration>
                  <source>1.6</source>
                  <target>1.6</target>
                  <excludes>
                      <exclude>**/*.*</exclude>
                  </excludes>
              </configuration>
           </plugin>
           <plugin>
              <groupId>org.codehaus.mojo</groupId>
              <artifactId>build-helper-maven-plugin</artifactId>
              <version>1.1</version>
              <executions>
                <execution>
                  <id>add-source</id>
                  <phase>generate-sources</phase>
                  <goals>
                      <goal>add-source</goal>
                  </goals>
                  <configuration>
                      <sources>
                          <source>src/groovy</source>
                          <source>src/java</source>
                      </sources>
                  </configuration>
                </execution>
              </executions>
           </plugin>
        </plugins>
  </build>
  <properties>
    <sonar.language>grvy</sonar.language>
    <sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>
    <sonar.surefire.reportsPath>test/reports</sonar.surefire.reportsPath>
    <sonar.cobertura.reportPath>target/test-reports/cobertura/coverage.xml</sonar.cobertura.reportPath>
    <sonar.phase>generate-sources</sonar.phase>
  </properties>
</project>
```

- Installation du plugin de couverture de test (http://grails.org/plugin/code-coverage).

Exécution

- cd /cytomine-web
- mvn sonar:sonar
- Go to serversonarip:9000

Utilisation

En Java:

- checkstyle: vérifie les règles de syntax (nom des méthodes,...): http://checkstyle.sourceforge.net/availablechecks.html
- PMD: détecte problèmes potentiels (code mort, code dupliqué, bugs possibles,...)

- Findbugs: détecte problèmes complexes (code vulnérable, mauvaises performances,...)
- Sonar squid: information sur le nombre de méthodes, classes,... + complexité
- Corbetura/clover/jacoco: couverture des tests