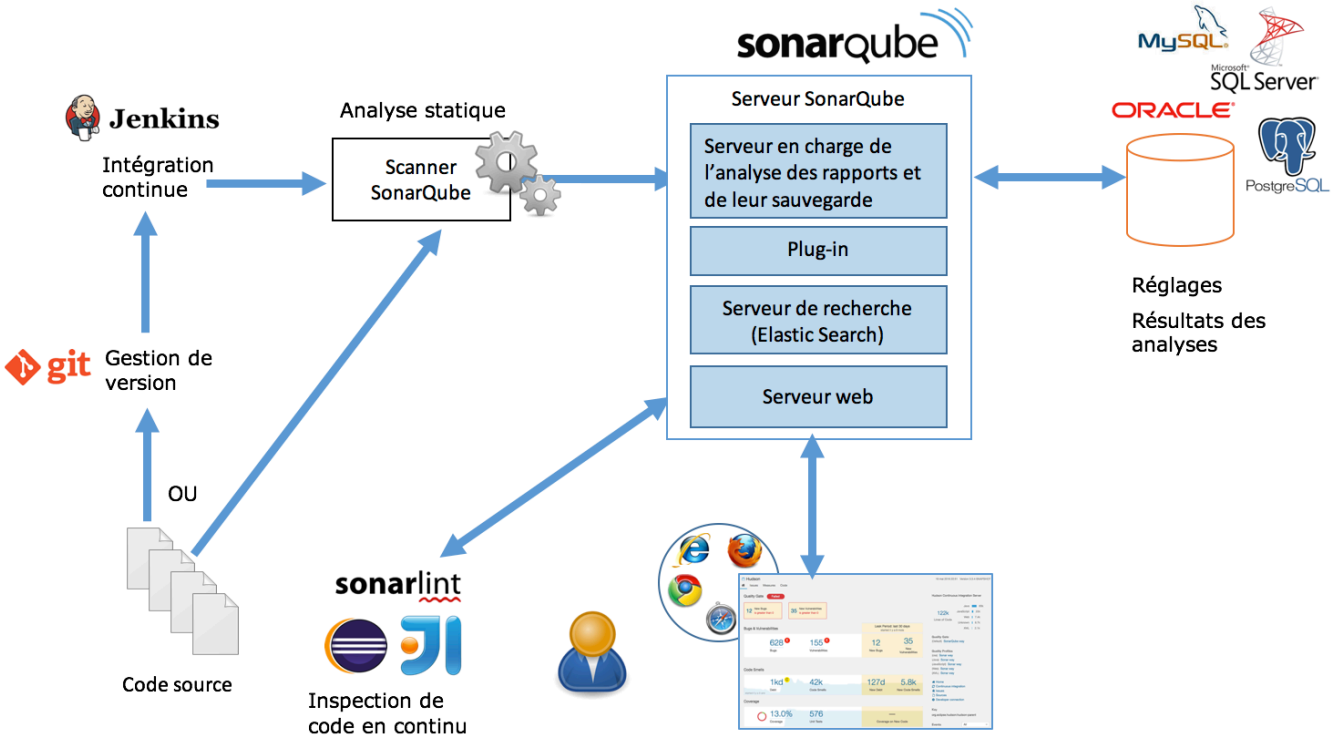
SonarQube

SonarQube est un logiciel *open source* de **mesure de la qualité du code source de projets de développement**. Il est développé par SonarSource, distribué sous licence GNU GPLv3. Il permet d'obtenir des informations sur la qualité au niveau du projet, du fichier ou d'un module et donne des indications sur chaque problème de qualité détecté et le temps de remédiation. Son périmètre est le **code source**, le **design** ainsi que les **tests unitaires**. Il supporte plus d'une **vingtaine de langages** de programmation, dont C/C++, C#, Java, Python, PHP, JavaScript et est traduit en une vingtaine de langues.

Pérennité : existe depuis 2007. Utilisé par la DSI du CNRS, le Ministère de la Défense, CISCO, Airbus, AirFrance, Boeing, BMW, Fiat, Renault, Volvo, SNCF, Banque Postale, BNP Paribas, Société Générale, MasterCard, Bouygues Telecom, Bosch, Amazon, ebay, PayPal, DHL, Oracle.

1) Architecture



SonarQube est une application web Java qui repose sur une base de données (par défaut H2, mais SonarQube peut utiliser MySQL, PostgreSQL, SQLServer, Oracle). Les résultats de l'analyse sont stockés dans la base de données, ce qui permet de **suivre dans le temps l'évolution de la qualité du projet**.

La plateforme SonarQube est constituée de plusieurs composants :

- un exécuteur qui lance les outils d'analyse de code sources externes et internes ;
- une soixantaine de plugins qui étendent SonarQube : support d'autres langages (PHP), métriques supplémentaires (couplage avec JDepend), utilisation d'analyseurs externes (PMD, Findbugs, …) ;
- un serveur Sonar qui :
  - agrège les résultats des analyses et les enregistre dans la base ;
  - permet aux développeurs de consulter les résultats des analyses des projets depuis un serveur web (tableaux de bords) ;
  - gère les recherches faites depuis l'interface web (ElasticSearch) ;
- une base de données pour stocker :
  - les réglages de configuration ;
  - l'historique des analyses des projets surveillés par sonar ;

Il est possible d'**automatiser l'analyse** avec un serveur **d'intégration continue** (ex. Jenkins, Travis).

SonarQube permet l'**inspection de code en continu** (intégration dans les IDE, vérification de la qualité depuis la dernière version ou la dernière analyse, notifications par email), ce qui permet de détecter les problèmes dès leur introduction dans le code, avant que le coût de remédiation soit élevé.

2) Qualité du code et mesures

SonarQube a été créé à l'origine pour agréger les résultats d'outils existants de contrôle de la qualité de code, pour évaluer la qualité globale ainsi que l'effort à fournir pour améliorer la qualité de l'application. Actuellement, le moteur SonarQube, et certains de ses plugins, réalisent une partie de l'analyse statique sans avoir recours à des outils externes, mais des plugins permettent de continuer à utiliser des outils externes (PMD, CheckStyle, …).

Analyses Java : SonarQube utilise les outils clover, cobertura (couverture des tests unitaires), google analytics, Squid for Java, Surefire (exécution de tests unitaires). Leur analyseur interne a remplacé checkstyle (règles de codage), JavaNCSS (métriques pour le code source), PMD (duplication de code, méthodes trop complexes, …) et findbugs. Il est encore possible de les utiliser via des plugin.

Analyses PHP : le plugin n'utilise que le moteur SonarQube, il n'est plus nécessaire d'installer PHPDepend, PHPCodeSniffer, PHPMD et PHPUnit.

SonarQube génère un rapport consultable dans un navigateur :

- densité des commentaires;
- taux de couverture des tests unitaires ;
- respect des conventions de nommage, des règles de codage et des bonnes pratiques ;
- détection de bogues ;
- détection de code mort ;
- détection de code dupliqué ;
- complexité du code (complexité cyclomatique, en 2017 la complexité cognitive devrait être introduite) ;
- score de maintenabilité, fiabilité et sécurité évalué à partir des résultats des analyses pour un profil qualité prédéfini ou personnalisé (jeux de règles).
- dette technique (estimation du temps nécessaire pour fixer tous les problèmes détectés dans le code, liés à la maintenabilité du code).

Sonar couvre les 7 axes de la qualité du code :

- architecture & design ;
- documentation ;
- respect des standards de codage ;
- non duplication du code ;
- tests unitaires ;
- complexité ;
- bogues potentiels.

Jusqu'à la version 5.5, SonarQube évaluait la dette technique en se basant sur la méthodologie SQALE (*Software Quality Assessment based on Lifecycle Expectations*, utilisé pour suivre la qualité du code des projets agiles). SonarQube la calculait à partir des problèmes (***issues***) détectés dans le code en utilisant un jeu de règles. Ce jeu devait comprendre les règles qui permettent d'obtenir des informations sur :

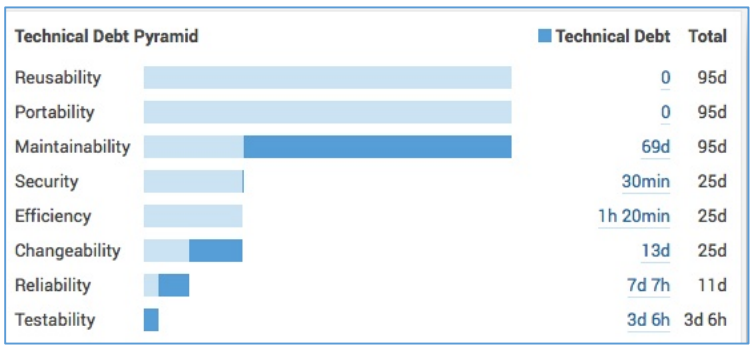
- la duplication de blocs,
- les tests unitaires qui ont échoué,
- le taux de couverture des branches des tests unitaires,
- le taux de lignes couvertes par les tests,
- la densité de commentaire.

Jusqu'à la version 5.5, la présentation des résultats était synthétisée dans une pyramide montrant la dette selon les 8 indices SQALE : réutilisabilité, portabilité, maintenabilité, sécurité, efficacité, changement, fiabilité, testabilité. Il fallait travailler

Dette technique

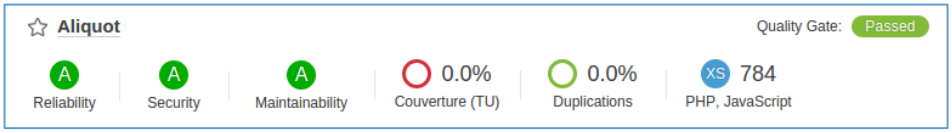
Lorsque le logiciel n'est pas basé sur les meilleurs pratiques possibles, une dette technique est introduite. Les intérêts ne cessent d'augmenter avec le temps (coûts supplémentaires dans le futur). Si le développeur code vite et mal, il contracte une dette technique qu'il faudra rembourser tout au long de la vie du projet (temps de développement de plus en plus longs, bugs de plus en plus fréquents).

La dette peut être non intentionnelle (non respect des règles de codages, il ne devrait jamais y avoir de telle dette) ou intentionnelle : la qualité augmente la charge de travail, pour livrer le projet dans les temps il faut parfois ne pas respecter une conception idéale. Il est conseillé dans ce cas de sortir une nouvelle version après la livraison, corrigeant au plus tôt la dette.



à l'amélioration du haut vers le bas : le code devait d'abord être testable, avant de pouvoir travailler sur l'amélioration de sa fiabilité, etc. Le temps de remédiation est indiqué en bleu pour chaque caractéristique, le temps total cumulé du bas vers le haut est indiqué en noir.

Depuis la version 5.6 (juin 2016), un tableau de bord montre les **scores** de maintenabilité, fiabilité et sécurité, le taux de couverture des tests, le taux de duplications, la taille du projet ainsi que les langages des sources.



1. Scores de maintenabilité, fiabilité et sécurité

Le calcul des scores est dépendant du profil qualité utilisé (jeux de règles) :

Score	Fiabilité	Sécurité	Maintenabilité (*)
A	0 bug	0 vulnérabilité	0.00 ≤ ratio dette technique ≤ 0.05
B	Au moins 1 bug mineur	Au moins 1 vulnérabilité mineure	0.06 ≤ ratio dette technique ≤ 0.1
C	Au moins 1 bug majeur	Au moins 1 vulnérabilité majeure	0.11 ≤ ratio dette technique ≤ 0.20
D	Au moins 1 bug critique	Au moins 1 vulnérabilité critique	0.21 ≤ ratio dette technique ≤ 0.5
E	Au moins 1 bug bloquant	Au moins 1 vulnérabilité bloquante	0.51 ≤ ratio dette technique ≤ 1
Issue	***Bug***	***Vulnerability***	***Code smell***

(°) Le **ratio de la dette technique** est le ratio entre le coût pour remédier aux problèmes de type *code smell* et le coût de développement de l'application. La formule de calcul est la suivante :

coût total de remédiation des *issues* / (coût pour développer une ligne de code x nombre de lignes de code)

L'idée est de déterminer si réécrire l'application est plus rentable que corriger tous les problèmes. Lorsque le ratio est trop grand, il est préférable de réécrire l'application de zéro plutôt que d'essayer de réduire la dette en corrigeant les problèmes.

NB : le nombre de minutes pour remédier à chaque *issue* est stocké dans la base de données.

2. Métrique de taille du code

- *Classes* = nombre de classes
- *Directories* = nombre de répertoires
- *Files* = nombre de fichiers
- *Lines* = nombre de lignes (nombre de retour chariot)
- *Lines of code* = nombre de lignes qui contiennent au moins un caractère qui n'est pas un espacement et qui ne fait pas partie d'un commentaire
- *Lines of code per language* = lignes de code non commentées par langage de programmation
- *Methods* = nombre de méthodes/fonctions
- *Projects* = nombre de projets
- *Statements* = nombre d'instructions (if, else, while, for, do, switch, break, continue, return, throw, finally, catch, …),

3. Mesures pour la documentation

- *Comment lines* = nombre de lignes contenant des commentaires de documentation ou du code commenté, les lignes de commentaires non significatives sont ignorées (ligne vide ou avec le caractère spécial *)
- *Comments (%)* = **densité de commentaire** calculée par la formule :

```
Comment lines / (Lines of code + Comment lines) * 100
```

  Densité de 50% = autant de lignes de code que de commentaires
  Densité de 100% = le fichier ne contient que des commentaires

- *Commented-out LOC* = nombre de lignes de code commentées.

4. Mesures pour la duplication

- *Duplicated blocks* = nombre de blocs de lignes dupliqués. Un bloc de code est considéré comme dupliqué s'il y a au moins 100 tokens successifs dupliqués sur 10 lignes de code (PHP, JavaScript). Pour un code Java il y a duplication s'il y a au moins 10 instructions successives dupliquées, quels que soient le nombre de tokens et de lignes.
- *Duplicated files* = nombre de fichiers impliqués dans des duplications
- *Duplicated lines* = nombre de lignes impliquées dans des duplications
- *Duplicated lines (%)* = **densité de duplication** calculée par la formule :

```
Duplicated lines / Lines * 100
```

5. Mesures pour les tests

- Nombre de tests unitaires (unit tests), nombre de tests unitaires qui ont échoué (unit test errors), nombre d'exception lors de l'exécution (unit test failures)
- nombre de lignes non couvertes (uncovered lines)
- nombre de conditions non couvertes (uncovered conditions)
- couverture : calcul =

  (CT + CF + LC) / (2*B + EL)

  CT : condition évaluées à *true* au moins une fois
  CF : conditions évaluées à *false* au moins une fois
  LC : lignes couvertes
  B : nombre total de conditions
  EL : nombre total de lignes exécutables

6. Mesures de complexité

- Complexity = Complexité totale du projet
- Complexity / method = Complexité moyenne par fonction
- Complexity / file = Complexité moyenne par fichier
- Complexity / class = Complexité moyenne par classe

Le calcul de la complexité d'une méthode se base sur la complexité cyclomatique. Elle est minimum de 1 (1 pour la méthode). Elle est incrémentée de 1 pour chaque if, case, for, while, throw, catch, return (sauf s'il est en dernière instruction de la fonction), &&, ||, opérateur conditionnel ternaire.

3) Installation de SonarQube (serveur + BD + application web)

Pré-requis côté serveur :

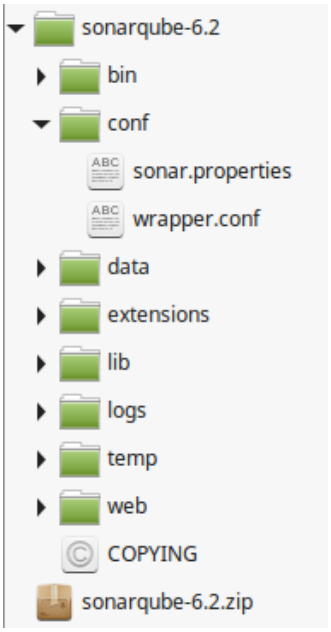
- Java : Oracle JRE 8 ou OpenJDK 8 (conseillé pour mac OS X)
- SGBD : MySQL ≥ 5.6 ou Oracle ≥ 11G ou PostgreSQL ≥ 8 ou Microsoft SQL Server ≥ 11.0
- RAM : 2 Go

Pré-requis côté client :

- navigateur web récent (fonctionne sous IE ≥ 11, Edge, Firefox, Chrome, Safari, SonarSource n'a pas fait de test sous Opera)
- JavaScript activé.

NB sur la VM exécuter les commandes ci-dessous pour supprimer l'ancienne version de MySQL et installer la 5.6 :

```
sudo apt-get purge mysql-server-5.5 mysql-client-5.5
sudo apt-get autoremove
sudo apt-get install mysql-server-5.6 mysql-client-5.6
```



1. Télécharger sonarqube 6.2 (décembre 2016)

https://www.sonarqube.org/downloads/

2. Configurer le fichier conf/sonar.properties pour qu'il crée le serveur web

```
# URL d'accès http://localhost:7223
# 0.0.0.0 = n'importe quelle interface réseau
sonar.web.host= 0.0.0.0
# port (valeur par défaut 9000)
sonar.web.port=7223
# si /sonar/ => URL = http://localhost:7223/sonar/
sonar.web.context=/
```

Décommenter les lignes indiquant le login/mot de passe de la base et leur mettre la valeur sonar

```
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
```

Décommenter la ligne pour MySQL :

```
#----- MySQL 5.x
sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterE
ncoding=utf8&rewriteBatchedStatements=true&useConfigs=maxPerformance
```

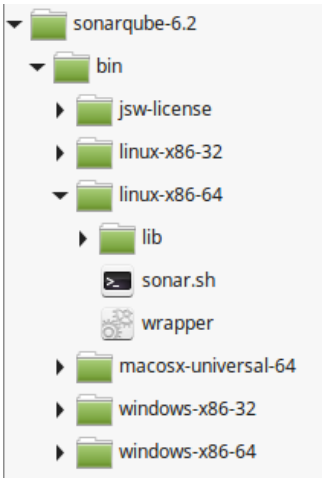
3. Créer la base MySQL *sonar* avec comme utilisateur *sonar* mot de passe *sonar*, lui donner les privilèges pour créer, mettre à jour et supprimer des objets dans le schéma. Le jeu de caractère doit être UTF-8.

```
use mysql ;
CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8_general_ci ;
CREATE USER 'sonar' IDENTIFIED BY 'sonar' ;
GRANT ALL ON sonar.* TO 'sonar'@'localhost ' IDENTIFIED BY 'sonar' ;
FLUSH PRIVILEGES ;
```

Vérifier en console qu'il est possible d'accéder à MySQL avec l'utilisateur sonar et le mot de passe sonar.

4. Lancer SonarQube

Aller dans le répertoire bin, entrer dans le sous-répertoire correspondant au système d'exploitation et CPU.



- Sous **linux/mac os** entrer la commande :

```
sudo ./sonar.sh start
```

Si le script sonar.sh de l'architecture x86-64 est exécuté et que le message d'erreur ci-dessous est affiché, c'est certainement un problème d'architecture, tester sonar.sh de la version x86-32

```
sonarqube-6.2/bin/linux-x86-64/./wrapper:
1:/home/jdev/Bureau/sonarqube-6.2/bin/linux-x86-
64/./wrapper: Syntax error: "(" unexpected
Failed to start SonarQube.
```

Vérifier si le système est 32 ou 64 bits :

```
getconf LONG_BIT
```

- Sous **windows x86-64** (ou 32 selon l'architecture), ouvrir une fenêtre de commande dans le répertoire, puis lancer le script StartSonar.bat

```
C:\Users\magali\Desktop\sonarqube-5.4\bin\windows-x86-32>StartSonar.bat
wrapper  | --> Wrapper Started as Console
wrapper  | Launching a JVM...
```

Le script lance SonarQube, le fichier de log logs/sonar.log doit contenir le message *SonarQube is up*.

En cas de problème (s'il n'y a pas de message « is up ») ou si le message ci-dessous est affiché :

wrapper | <-- Wrapper Stopped

Ouvrir le fichier **sonar.log** dans le répertoire logs, il contient les traces expliquant pourquoi sonar ne démarre pas.

C'est généralement un problème de droits ou d'accès à mysql, si le root mysql n'a pas de mot de passe et que la table *user* de la base *mysql* contient une ligne où le champ User est vide, alors il est impossible de se connecter avec un mot de passe pour les autres utilisateurs).

Ex de trace d'erreur dans sonar.log :

```
Caused by: org.apache.commons.dbcp.SQLNestedException: Cannot create PoolableConnectionFactory (Access denied for user
'sonar'@'localhost' (using password: YES))
```

La base contient un utilisateur vide sans mot de passe, il faut supprimer cette ligne de la table *user* de la base *mysql*, faire un *flush privileges* et réessayer d'accéder en console à mysql avec l'utilisateur *sonar* et le mot de passe *sonar*.

```
use mysql ;
select Host, User, Password from user;
```

```
+-----------+-------+-----------
| Host      | User  | Password
+-----------+-------+-----------
| localhost | root  |
| linux     | root  |
| localhost |       |
```

```
delete from user where User='';
flush privileges;
```

Sous linux un autre problème peut être rencontré (sonar.log) :

```
OpenJDK Client VM warning: You have loaded library /home/jdev/Bureau/sonarqube-6.2/bin/linux-x86-32/lib/libwrapper.so which might have disabled stack guard. The VM will try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>', or link it with '-z noexecstack'.
```

Pour corriger le problème, aller dans sonarqube-6.2/bin/linux-x86-32, exécuter les commandes :

```
sudo apt-get install execstack
execstack -c lib/libwrapper.so
```

Si la version de Java n'est pas la 8 un message apparaît dans les logs de Sonar :

```
WrapperSimpleApp: Unable to locate the class org.sonar.application.App: java.lang.UnsupportedClassVersionError: org/sonar/application/App : Unsupported major.minor version 52.0
```

Vérifier la version de Java :

```
java -version
```

Installer Java 8 :

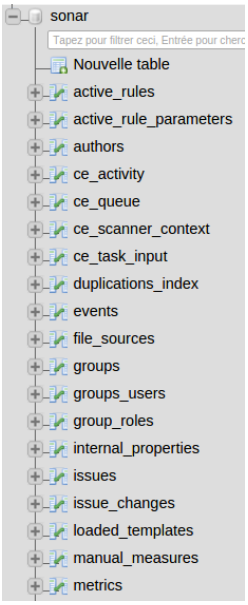
```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

NB : accepter la licence

```
sudo apt-get install oracle-java8-set-default
java -version
```

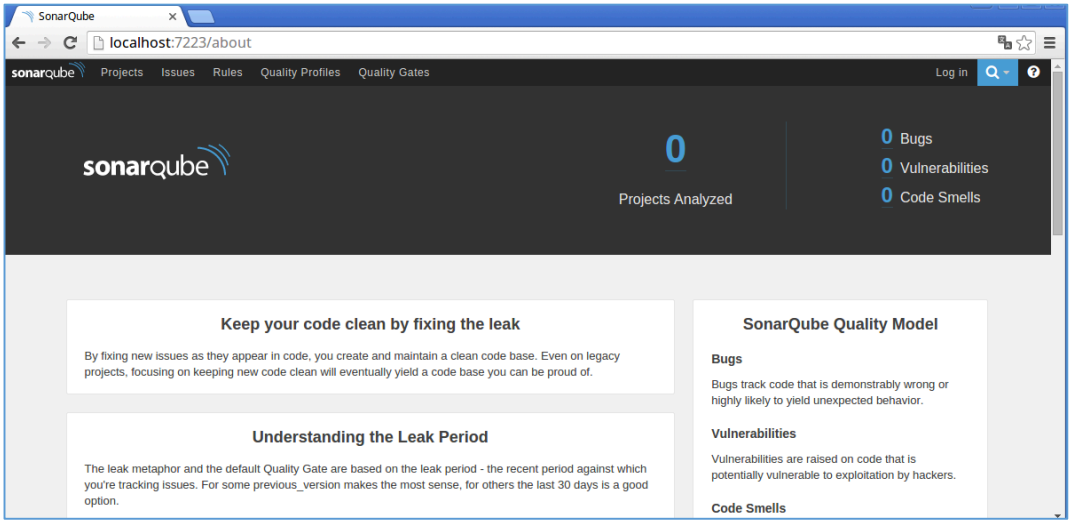
Si le message obtenu est *Process[web] failed to start*
Vérifier qu'il n'y a pas d'accents dans le chemin du répertoire de sonarqube.
Regarder l'erreur dans le fichier web.log

La première fois que sonar est lancé, il crée, les tables pour stocker les analyses dans la base *sonar* de MySQL.



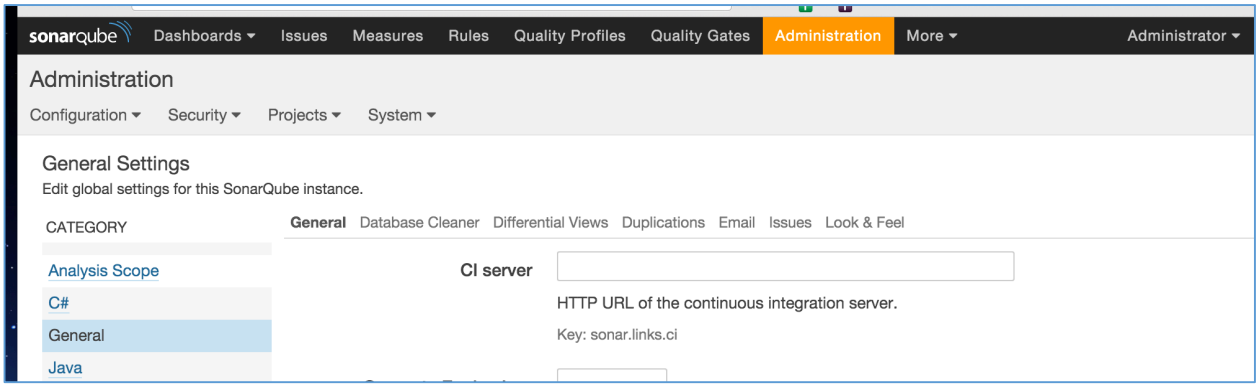
Le serveur web Sonar écoute sur le port **7223**.

5. Aller dans le navigateur pour accéder à l'**interface web de sonarQube** :
http://localhost:7223/

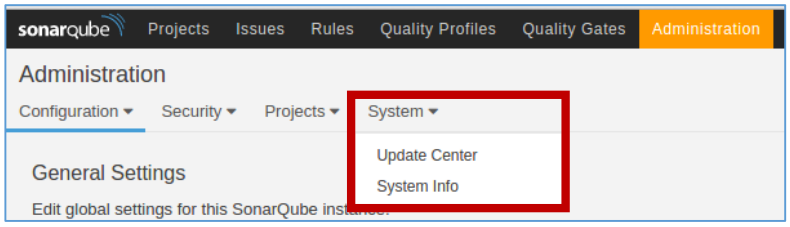


4) Configuration

1. Cliquer à droite de l'écran sur « Log in » pour se connecter comme **administrateur** (compte par défaut admin/admin), puis cliquer sur « Administration » dans le menu horizontal

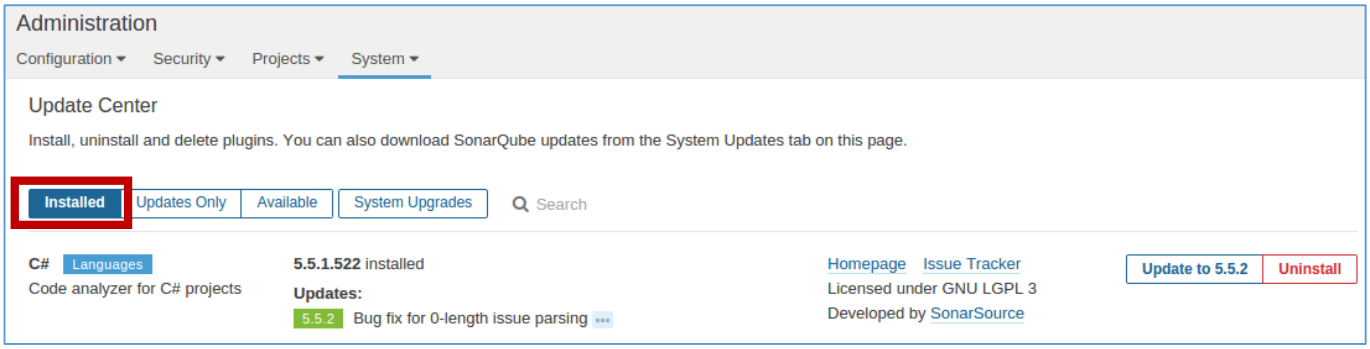


2. Ajouter ou mettre à jour des plugins

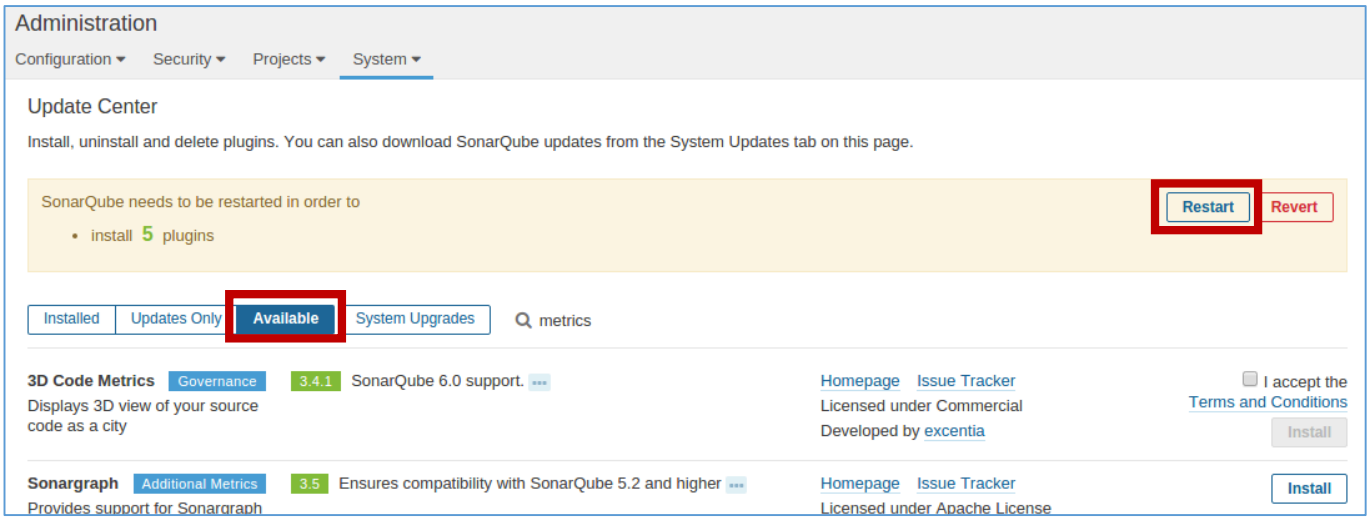


Dans le menu d'administration sélectionner *System* > *Update Center*.

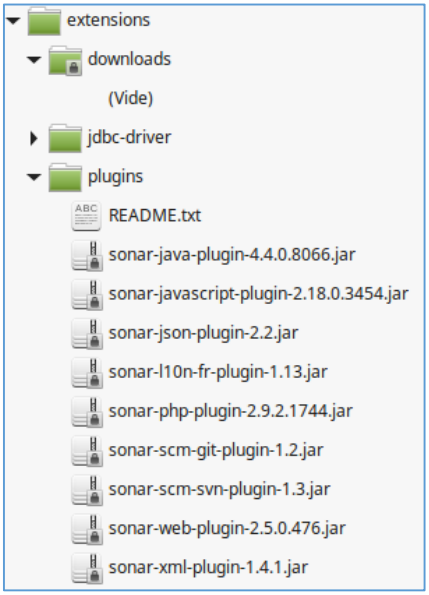
L'onglet *Installed*, du centre de mises à jour, liste les plugins installés. Il permet d'en désinstaller ou de mettre à jour ceux déjà installés (il faut redémarrer le serveur pour que la nouvelle version d'un plugin soit prise en compte).

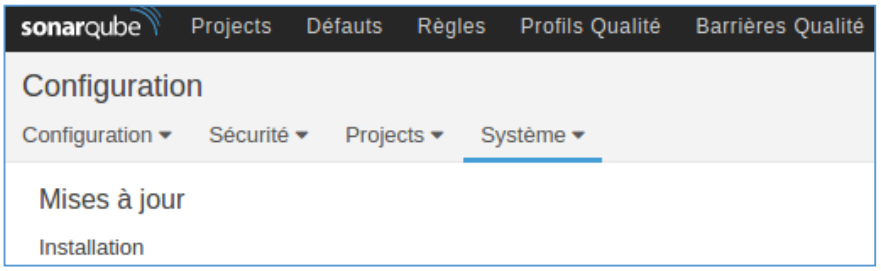


Aller dans l'onglet *Available* pour installer les plugins souhaités, par exemple French Pack, JSON, PHP, Web, XML. Relancer sonar en cliquant sur le bouton « Restart » en haut de la page du centre de mises à jour.



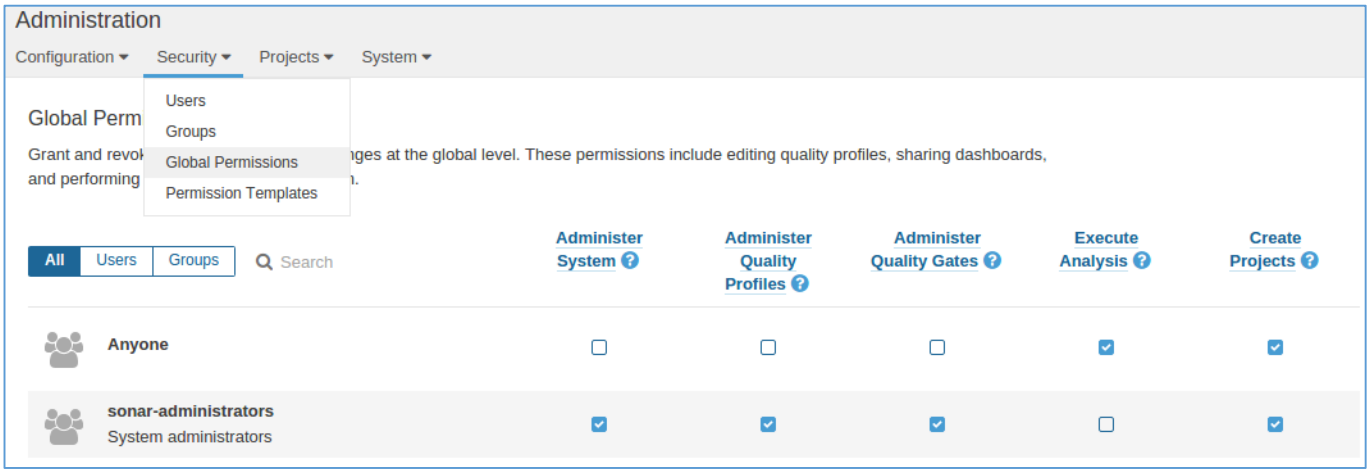
Les plugins s'installent dans le sous-répertoire plugin du répertoire extension. Si le plugin français (sonar-l10n-fr-plugin) est installé, l'interface est en français





3. Sécurité

L'onglet Security du menu d'administration permet de gérer l'authentification et les autorisations : création d'utilisateurs et de groupes, permissions d'accès aux composants et aux informations. Il est possible de restreindre la visibilité d'un projet aux utilisateurs d'un groupe, de définir les utilisateurs ou groupes autorisés à administrer un projet, …



L'authentification et les autorisations peuvent être déléguées à un système d'authentification externe grâce à des plugins : github, LDAP, …

4. Paramètres généraux

Le menu *Configuration > General Settings > General* permet de :

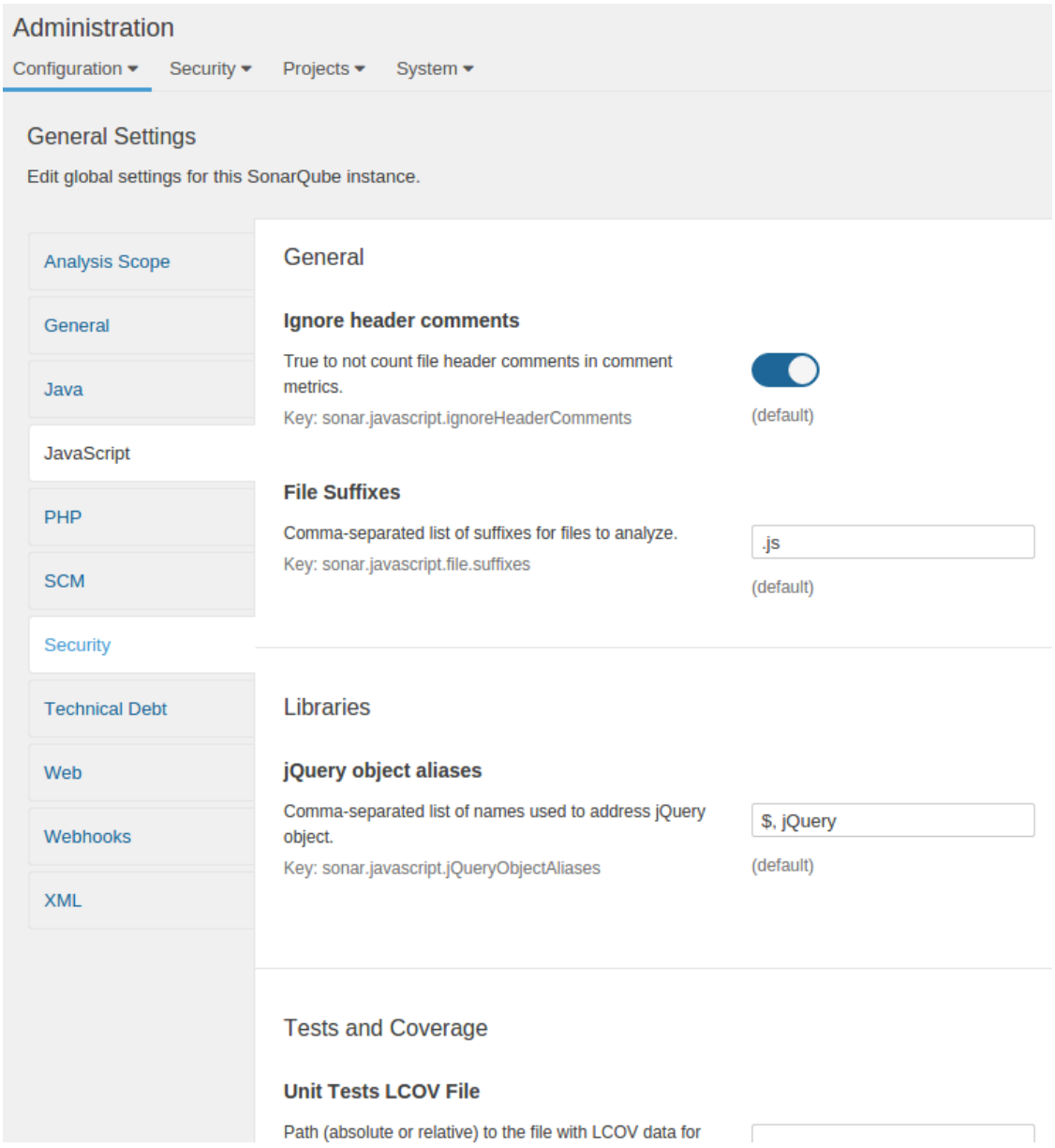
- Définir l'adresse d'un serveur d'intégration continue
- Définir les informations pour l'envoi de notifications par email (les mails sont envoyés aux utilisateurs qui souscrivent aux notifications, l'administrateur ne peut pas définir qui en reçoit)
- Activer/désactiver la détection de duplication de code entre projets (par défaut la duplication de code est recherchée uniquement dans le projet)
- Déterminer les analyses à conserver (ex. un snapshot par semaine s'il y en a eu plusieurs dans la même semaine, supprimer les snapshots après un nombre de semaines, …)

5. Paramètres d'analyse

Le menu *Configuration* > *General Settings* > *Analysis Scope* permet d'exclure des fichiers de l'analyse et d'ignorer des *issues*.

Des réglages peuvent être réalisés pour chaque plugin d'analyse d'un langage à partir du menu *Configuration* > *General Settings* :

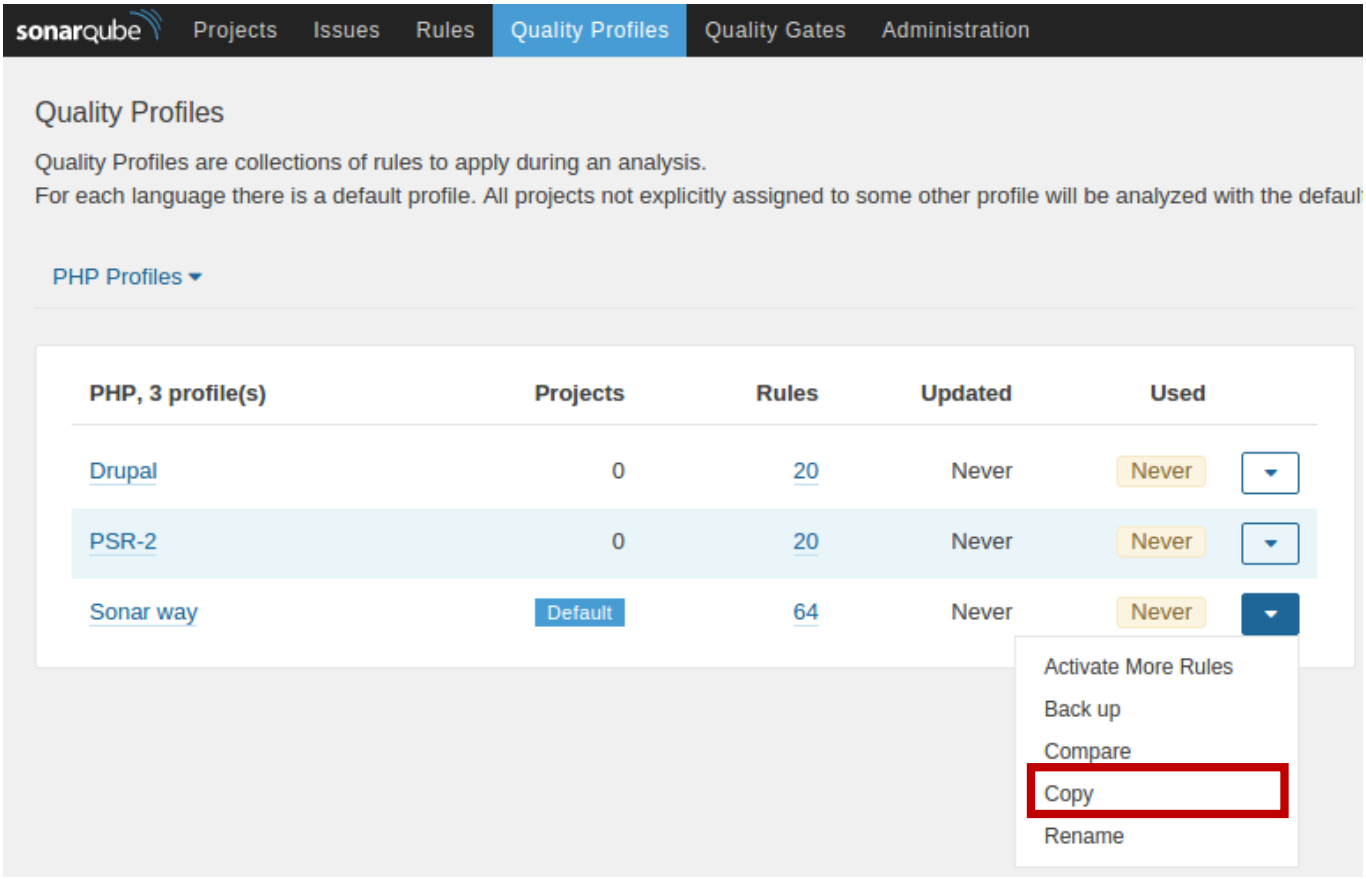
- Extension des fichiers qui seront analysés par le plugin
- Emplacement des fichiers de couverture des tests, des rapports d'exécution des tests unitaires



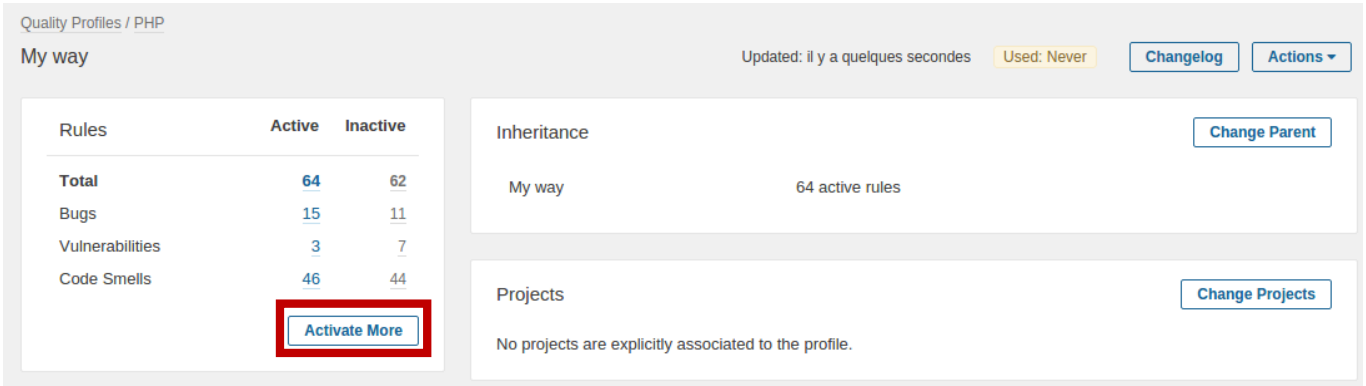
6. Créer et personnaliser un profil qualité

Un profil qualité comporte un jeu de règles qui seront utilisées pour l'analyse d'un projet. Les profils permettent d'analyser des applications avec des exigences plus ou moins fortes. L'onglet *Quality Profiles* permet de définir les règles à utiliser pour un profil. Les profils sont listés par langage, il existe au moins un profil par langage (Sonar way). Par exemple, trois profils sont définis pour le langage PHP, celui utilisé par défaut est Sonar way, il comporte 64 règles.

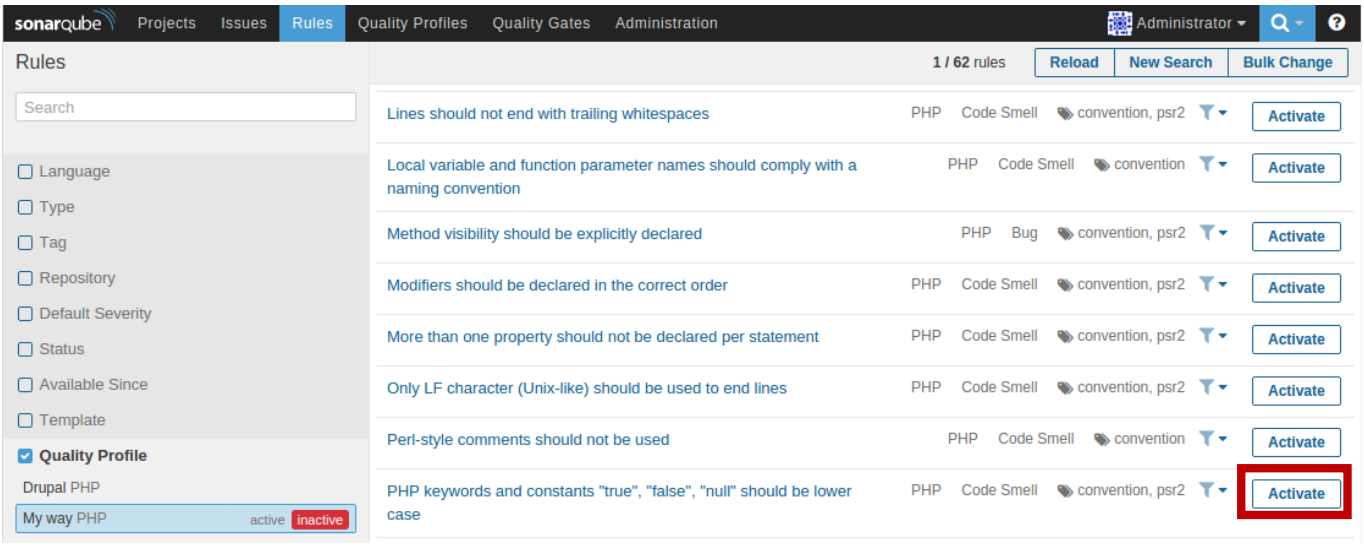
Créer un nouveau profil PHP en copiant celui de Sonar way.



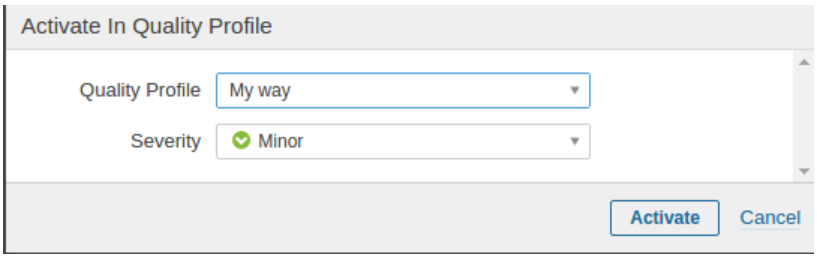
Le profil copié peut être affecté à des projets dans le bloc *Projects*. Le bloc *Rules* indique le nombre de règles activées pour ce profil, classées par types. Le bloc *Inheritance* permet d'hériter des règles d'un autre profil (par exemple le profil général utilisé par une entreprise).



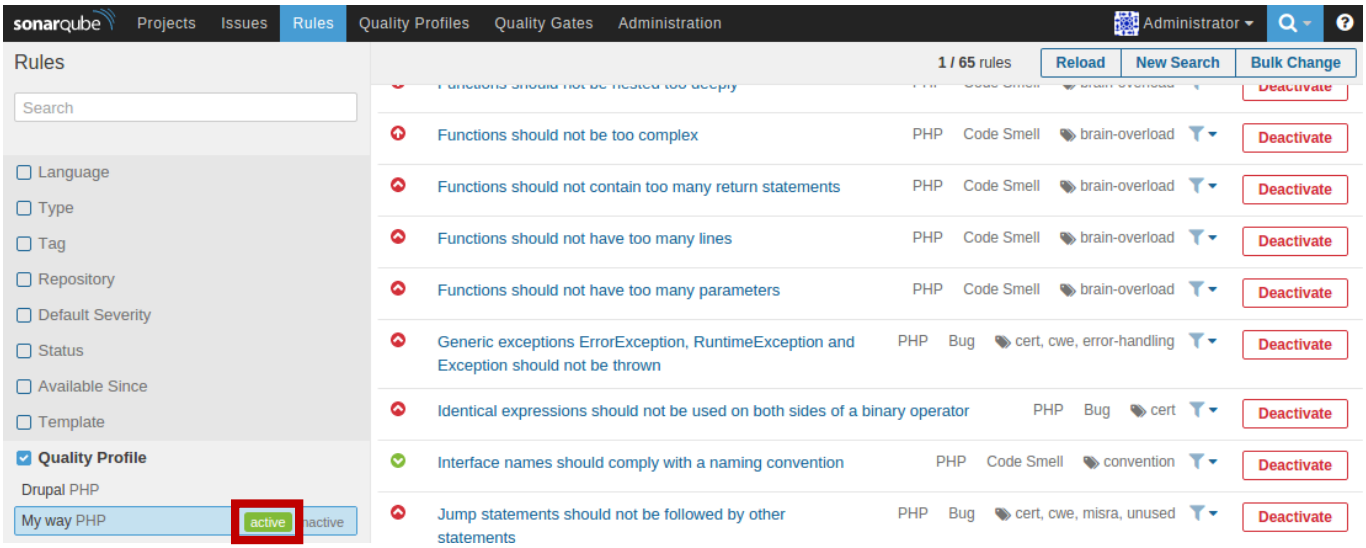
Un clic sur *Activate More* affiche la page de l'onglet *Rules* du menu d'administration, cette dernière permet d'ajouter ou supprimer des règles à un profil.



Un clic sur *Activate*, à droite de la règle, affiche le profil qualité auquel affecter la règle ainsi que le niveau de sévérité à associer à la règle pour ce profil :



Pour désactiver des règles, cliquer sur « active » dans le cadre de gauche à côté du nom du profil qualité, la liste des règles actives pour le profil est affichée :

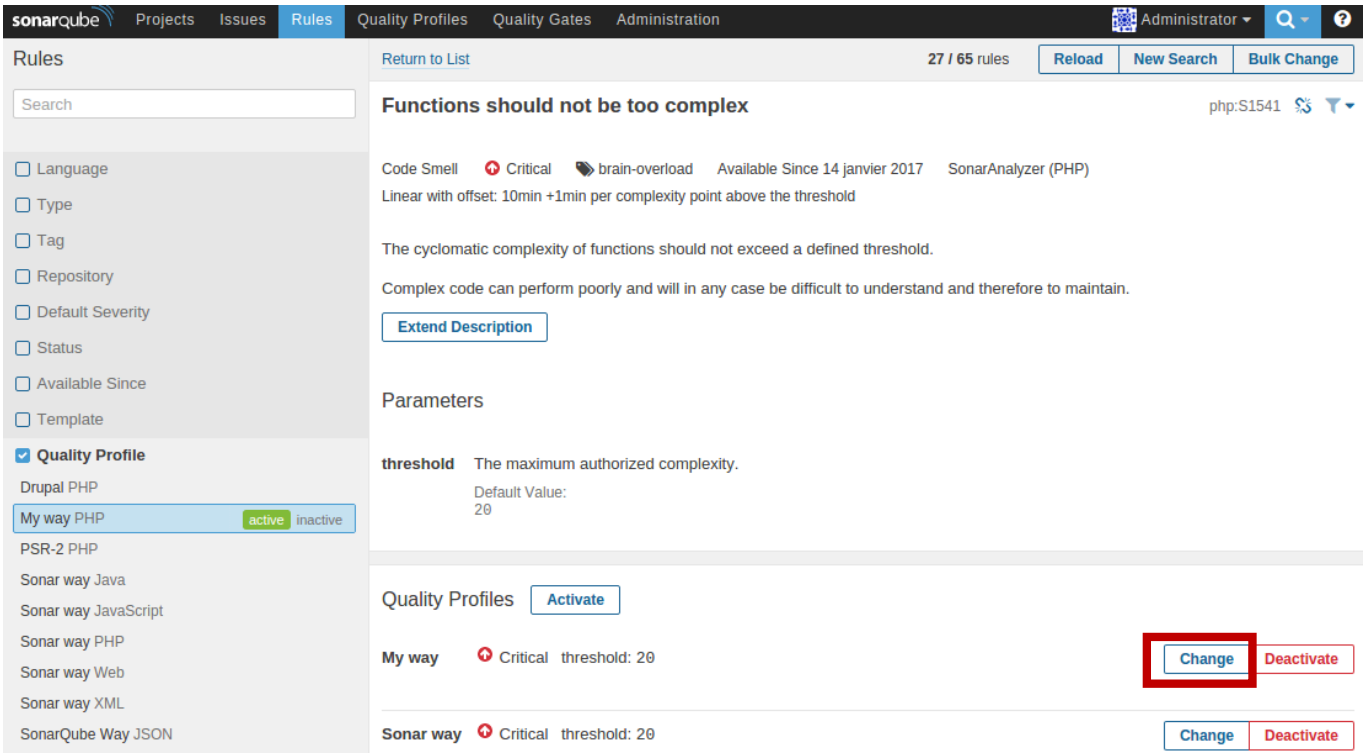


Modifier le seuil d'une règle

Un clic sur une règle affiche des informations et permet de modifier ses paramètres.

Les informations affichées sont les suivantes :

- explication sur la mesure utilisée (complexité cyclomatique)
- date d'ajout de la règle dans le plugin
- type : bug, vulnerability, code smell
- niveau de sévérité du problème : blocker , critical, major, minor , info.
- tag indiquant la classe du problème : brain overload, unused, convention, suspicious, …
- langage auquel la règle s'applique
- paramètres : seuil qui déclenche le problème



Un clic sur *Change* permet de modifier la valeur d'un paramètre et son niveau de criticité pour le profil choisi.

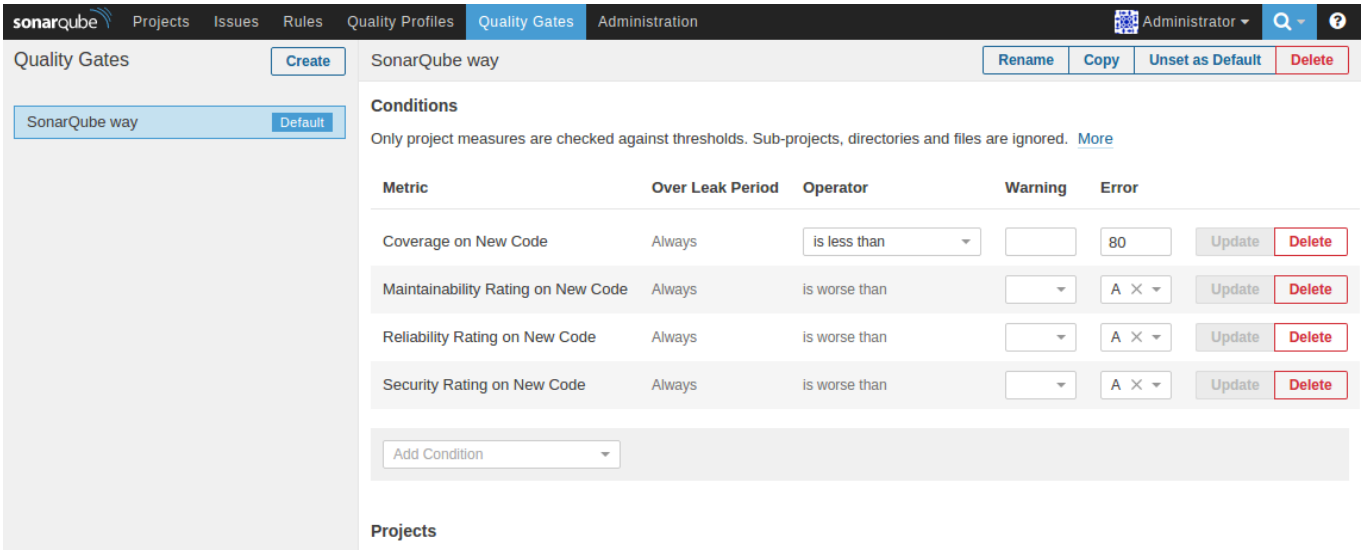
Utiliser par défaut le nouveau profil

7. Barrières qualité

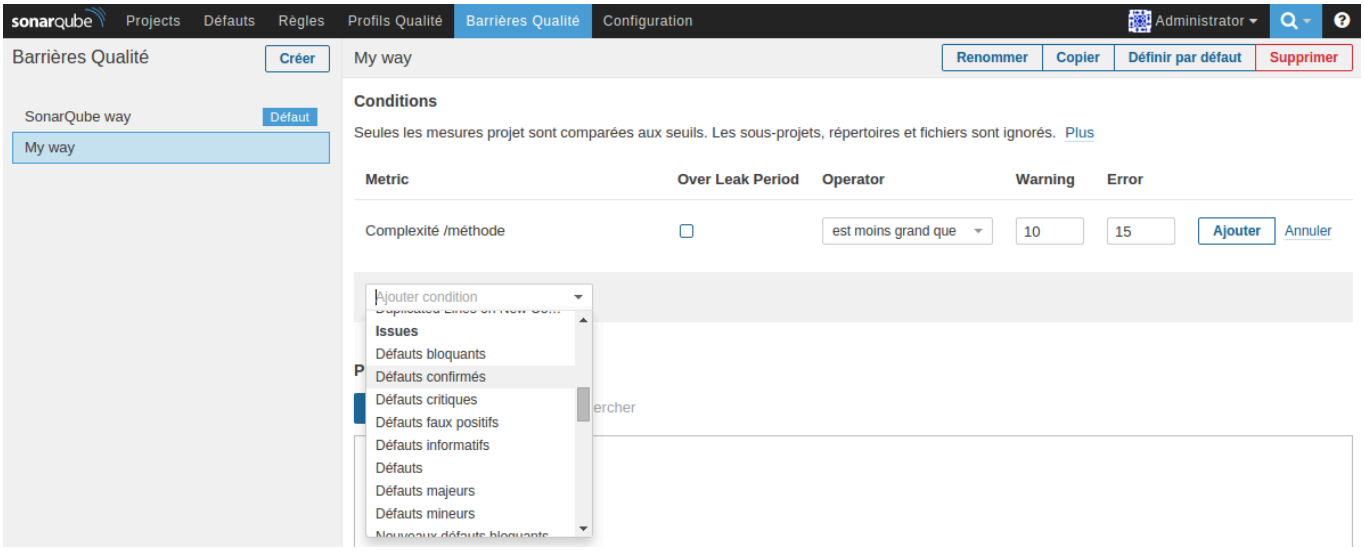
La page *Quality gates* permet de définir les exigences qu'un projet doit satisfaire pour être mis en production. Il est composé d'un ensemble de conditions booléennes.

SonarQube propose un profil par défaut. Les exigences pour ce profil sont :

- taux de couverture des tests ≥ 80
- score de maintenabilité : A
- score de fiabilité : A
- score de sécurité : A



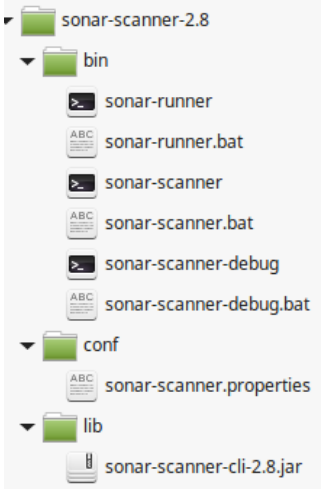
Il est possible de créer ses exigences en ajoutant des conditions à une nouvelle barrière qualité :



La nouvelle barrière peut s'appliquer à un ou plusieurs projets.

SonarQube affiche pour chaque projet s'il est prêt à être mis en production (*passed*) ou non (*failed*) au regard des conditions définies pour la barrière qualité :

5) Installation de SonarScanner (scanner de sources en ligne de commande)



1. Télécharger SonarQube Scanner 2.8

2. Modifier le fichier conf/sonar-scanner.properties

```
sonar.host.url=http://localhost:7223
sonar.sourceEncoding=UTF-8
```

3. Ajouter le chemin du répertoire bin de sonar-scanner au PATH

```
export PATH=$PATH:chemin vers sonar scanner
```

4. Vérifier l'accès à la commande en ouvrant une nouvelle console et en tapant :

```
sonar-scanner -h            linux/mac os
sonar-scanner.bat -h        windows
```

5. Dans le projet à analyser (celui indiqué dans *sonar.sources*) créer à la racine un fichier *sonar-project.properties* pour configurer l'analyse.

```
# cle unique
sonar.projectKey=IBDM:aliquot
# nom et version du projet affichés par l'interface graphique
sonar.projectName=Aliquot
sonar.projectVersion=1.0
# chemin relatif des fichiers sources, séparateur de chemins = virgule
sonar.sources=src/AppBundle,web/js
# encodage des fichiers sources
sonar.sourceEncoding=UTF-8
```

6. Se placer à la racine du projet et lancer le scanner (si la variable d'environnement n'est pas configurée, lancer la commande depuis ce répertoire en utilisant le chemin complet de sonar-scanner)

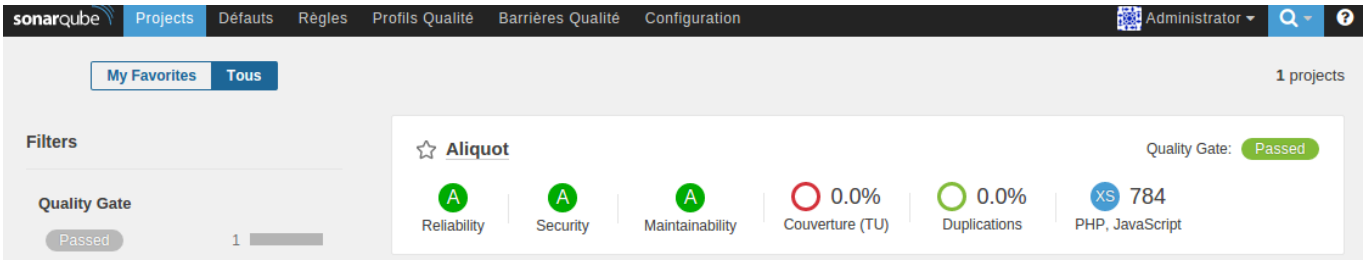
```
sonar-scanner               (sous windows sonar-scanner.bat)
```

```
INFO: ANALYSIS SUCCESSFUL, you can browse http://localhost:7223/dashboard/index/IBDM:aliquot
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted an
alysis report
INFO: More about the report processing at http://localhost:7223/api/ce/task?id=AVmeJCebhvROrzmlykwx
INFO: Task total time: 3.281 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 4.432s
INFO: Final Memory: 45M/109M
INFO: ------------------------------------------------------------------------
```

Les résultats sont placés dans la base de données

	id	name	description	enabled	scope	qualifier	kee
Modifier Copier Effacer	1	Aliquot	*NULL*	1	PRJ	TRK	IBDM:aliquot

7. Aller dans l'onglet *Projects* de l'interface web pour voir les résultats de l'analyse

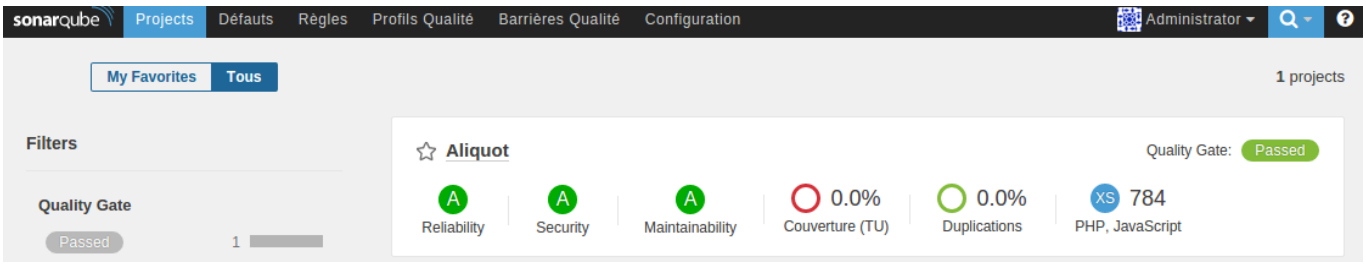


NB : l'analyse peut être lancée en ligne de commande, ou automatiquement par Maven, Jenkins, …
SonarQube conserve un historique des analyses d'un projet.

6) Consulter les résultats de l'analyse dans l'interface web

L'analyse de SonarQube fournit des mesures de la qualité du code ainsi que des *issues* pour chaque règle qui n'est pas satisfaite.

7. Tableau de bord des projets dans l'onglet *Projects*



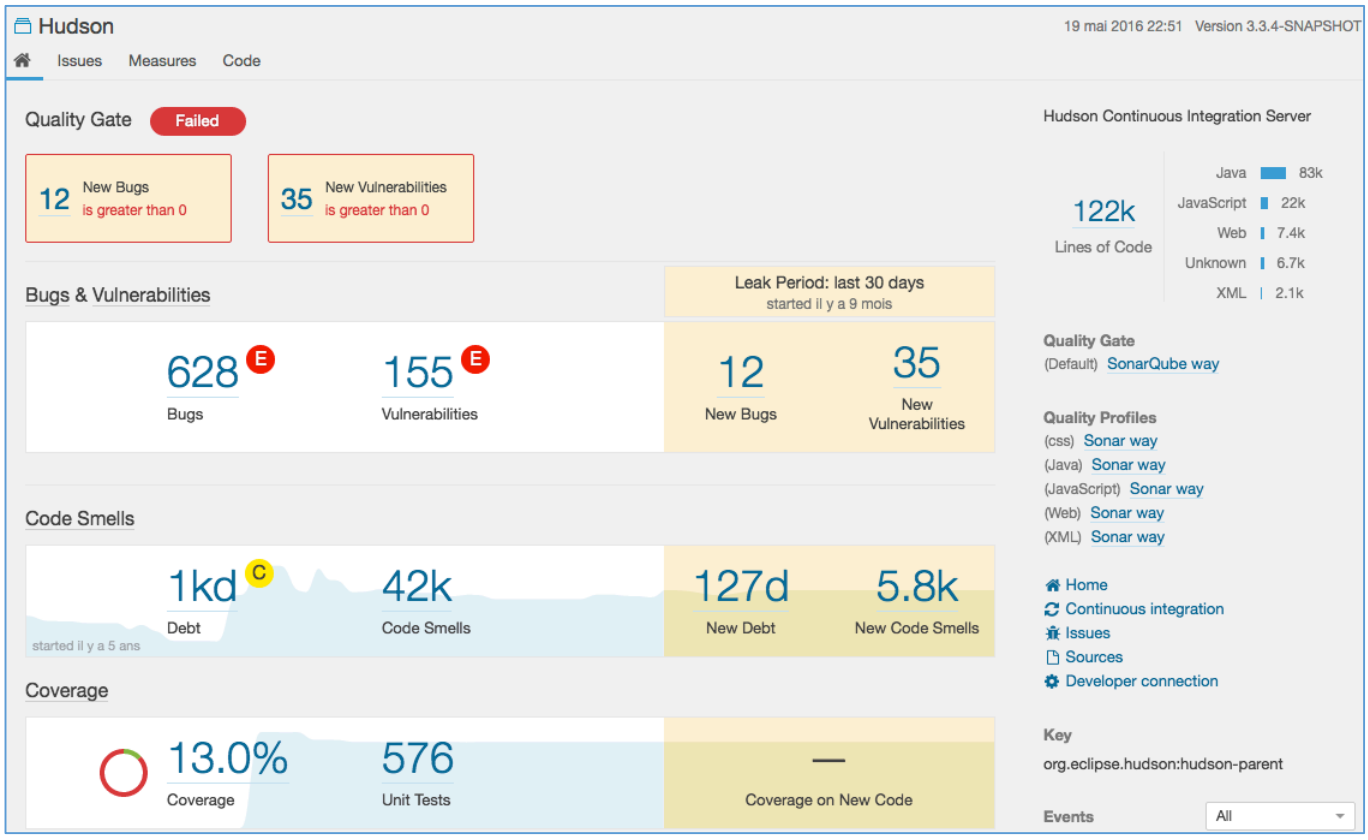
Les informations disponibles sont les suivantes :

- nom du projet,
- le projet est prêt à être mis en production (passed) ou non (failed),
- taille en Kloc avec une indication de la taille du projet (XS, S, M, L, XL),
- langages utilisés dans le projet,
- évaluation de la fiabilité, sécurité, maintenabilité (score de A à E),
- taux de couverture,
- le taux de duplication.

8. Page du projet

Un cic que le nom du projet, dans le tableau de bord, affiche la page du projet, elle indique:

- si le projet est prêt à être mis en production (quality gate),
- la qualité du projet et l'évaluation de la dette technique,
- la qualité de ce qui a été produit depuis la version précédente (leak period, cette période peut être modifiée dans l'onglet *Administration* > *Configuration* > *General Settings* > *General*).

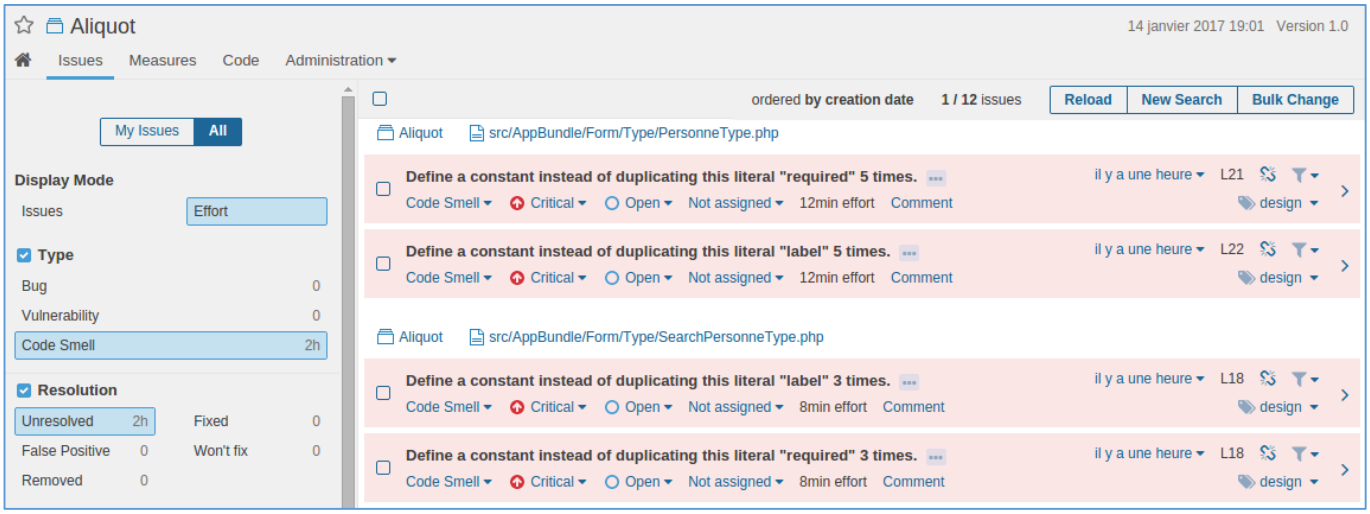


Pour améliorer la qualité du projet la priorité est de **résoudre les nouveaux problèmes** (analogie de la fuite d'eau : fermer le robinet avant d'éponger, ie le but est de ne pas introduire de nouveaux problèmes de qualité, de plus il est plus facile d'intervenir sur du code frais que sur un ancien code). Il faut corriger en priorité ce qui apparaît dans la partie droite sur fond jaune. Intervenir sur la duplication de code, puis corriger les bugs et code smells.

Un clic sur une mesure, la dette, etc affiche plus d'informations.

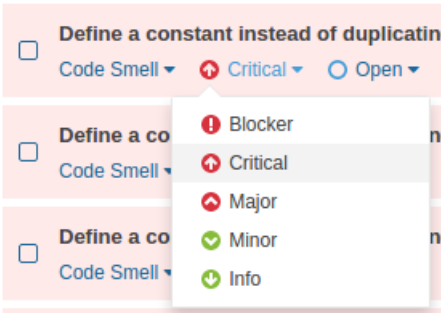
9. Liste des problèmes (*issues*)

L'onglet *Issues* affiche les problèmes, la durée pour les résoudre et des informations sur chaque problème : niveau de sévérité, type, tag.



Type de l'issue (modifiable) :
- Bug : code de case identiques dans un switch, boucle infinie, …
- vulnerability : problème de sécurité (mot de passe en clair, cookie secure, …)
- code smell : problème de conception ou de design (complexité cyclomatique, lignes de code commenté, …)

Niveau de sévérité du problème (il peut être modifié) :
- blocker : bug qui a une forte probabilité d'impacter le comportement, à régler immédiatement
- critical : problème qui peut impacter le comportement ou problème de sécurité (injection SQL)
- major : problème impactant fortement la productivité du développeur : duplications, paramètres non utilisés, …
- minor : problème impactant faiblement la productivité du développeur : lignes trop longues, …
- info.



Il est possible de marquer chaque *issue* :
- confirm : confirmer qu'il y a un problème, le statut passe de Open à Confirmed
- false positive : ce n'est pas un vrai problème
- won't fix : c'est un problème valide mais qui ne nécessite pas d'être modifié
- change severity : c'est un problème à régler mais il n'a pas le niveau de sévérité indiqué
- resolve : le problème a été réglé, lors de la prochaine analyse, si c'est vraiment le cas l'issue prendra le statut Closed, sinon son statut sera ré-ouvert.

Le problème peut être affecté à un utilisateur enregistré de SonarQube, un commentaire peut être ajouté (Comment).

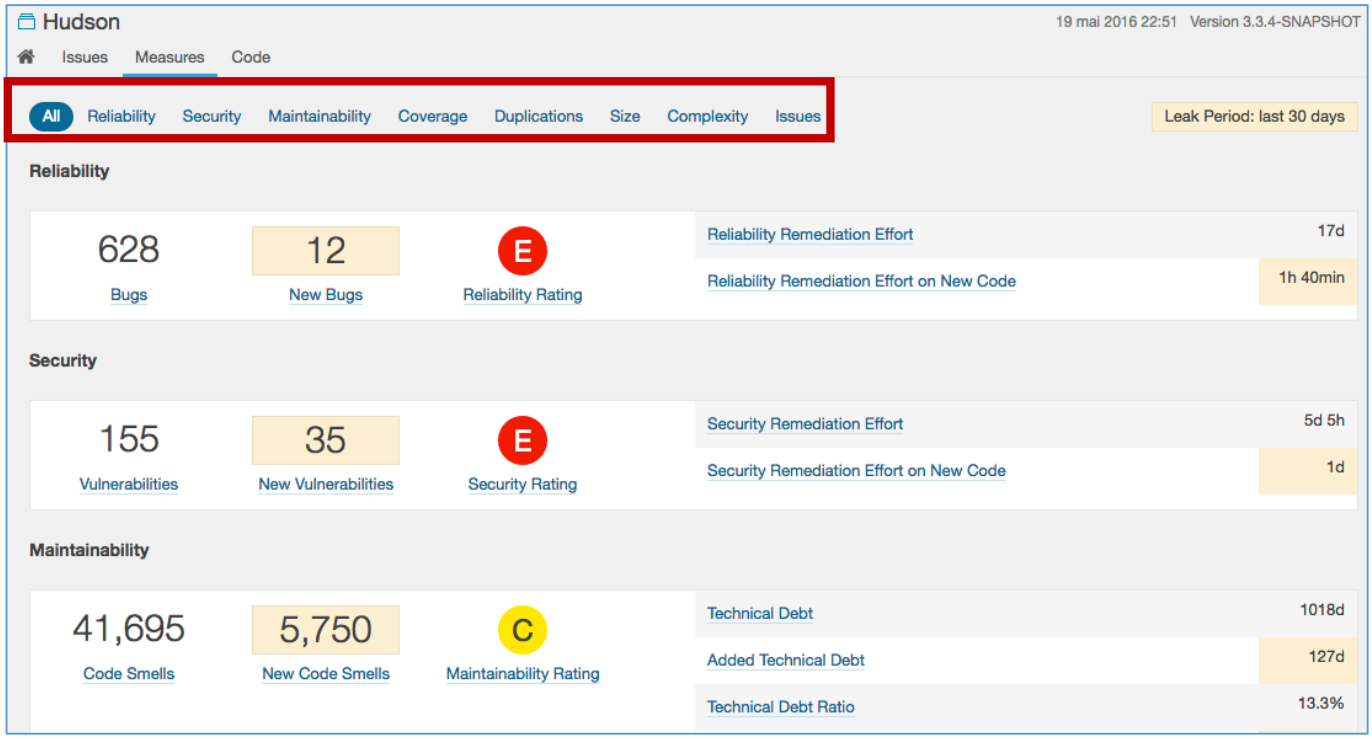
Lorsque beaucoup d'issues sont marquées *false positive* ou *won't fix*, il faut changer les règles du jeu utilisé car elles ne sont pas appropriées pour le projet (désactiver des règles dans le profil qualité ou modifier les paramètres).

Tag indiquant la classe du problème. Exemple :

- brain overload : trop d'information à garder en mémoire pour comprendre le code
- cert : règle du standard CERT (bonnes pratiques de programmation)
- clumsy : pourrait être accompli de manière plus concise ou claire
- confusing : la compréhension prendra du temps
- convention : convention de codage (formatage, nom, espaces, …)
- cwe : règle du CWE (Common Weakness Enumeration)
- misra : règle du standard MISRA (Motor Industry Software Reliability Association)
- owasp-* : règle de l'OWASP Top Ten
- psr2 : standard de codage PHP
- san-top25-* : SANS Top 25 coding errors
- security
- suspicious
- unpredictable : le code pourrait avoir un comportement imprévisible si les conditions changent
- unused : code non utilisé

10. Mesures

L'onglet *Measures* affiche des informations sur les mesures, la dette technique

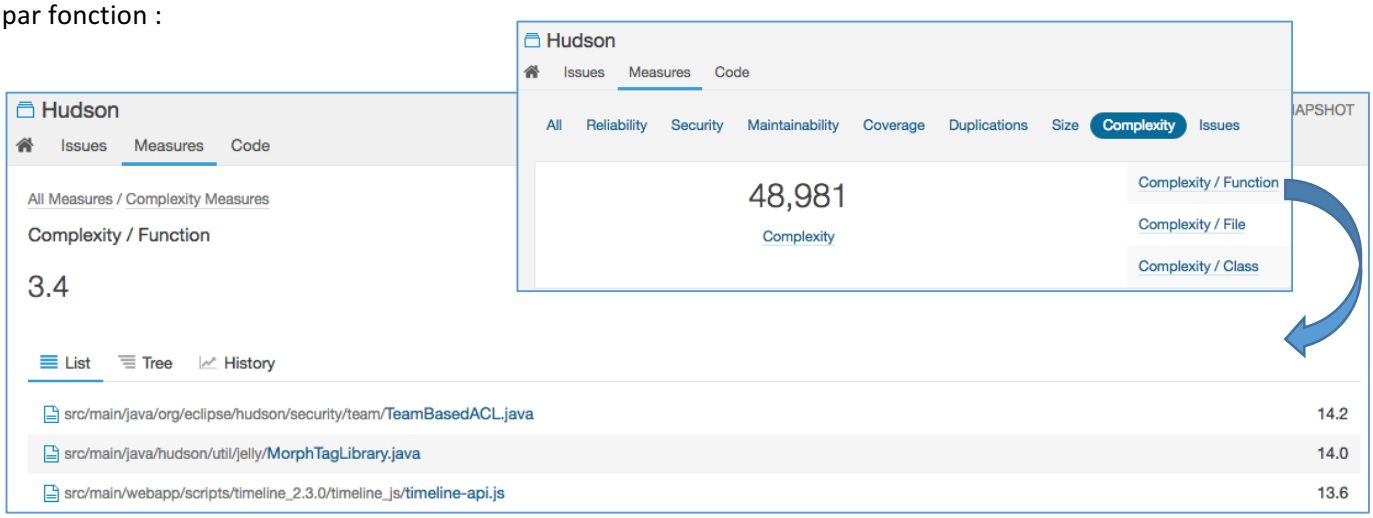


Un clic sur une mesure affiche la liste des fichiers concernés, il est possible de consulter le code du fichier.

Le menu *Measures* permet de visualiser les mesures par type :

- **Fiabilité** : score, nombre de bugs, effort de remédiation,
- **Sécurité** : score, nombre de vulnérabilités, effort de remédiation,
- **Maintenabilité** : score, nombre de code smells, dette technique, effort pour obtenir le score A,
- **Couverture** : taux de couverture, nombre de tests unitaire, informations quantitatives (nombre de lignes non couvertes, nombre de conditions non couvertes, nombre d'échecs de tests unitaires, …),
- **Duplication** : taux de duplication, nombre de blocs/lignes/fichiers dupliqués,
- **Taille du code** : nombre de lignes, d'instructions (if, else, while, for, do, switch, break, return, throw, finally, catch, …), de fonctions, classes, fichiers, répertoires, lignes commentées (en excluant les lignes non significatives vides ou contenant uniquement des caractères *), densité de documentation (50% = autant de lignes de code que de commentaires, 100% = le fichier ne contient que des commentaries) ,
- **Complexité** : complexité totale, complexité moyenne par fonction / fichier / classe ;

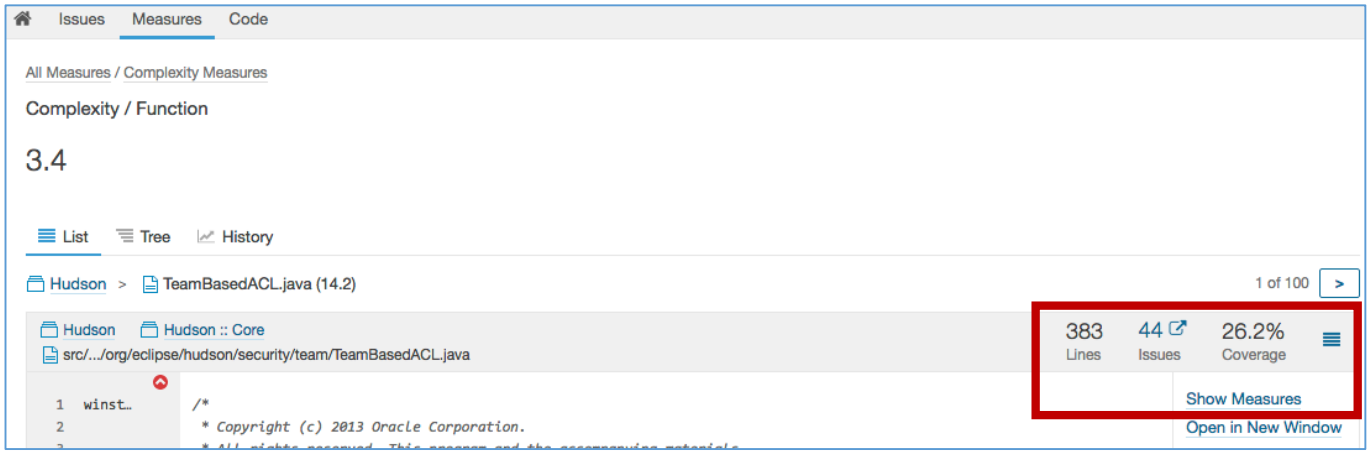
Un clic sur *Complexity* / *Function* affiche la liste des fichiers, classés par ordre décroissant de complexité moyenne par fonction :



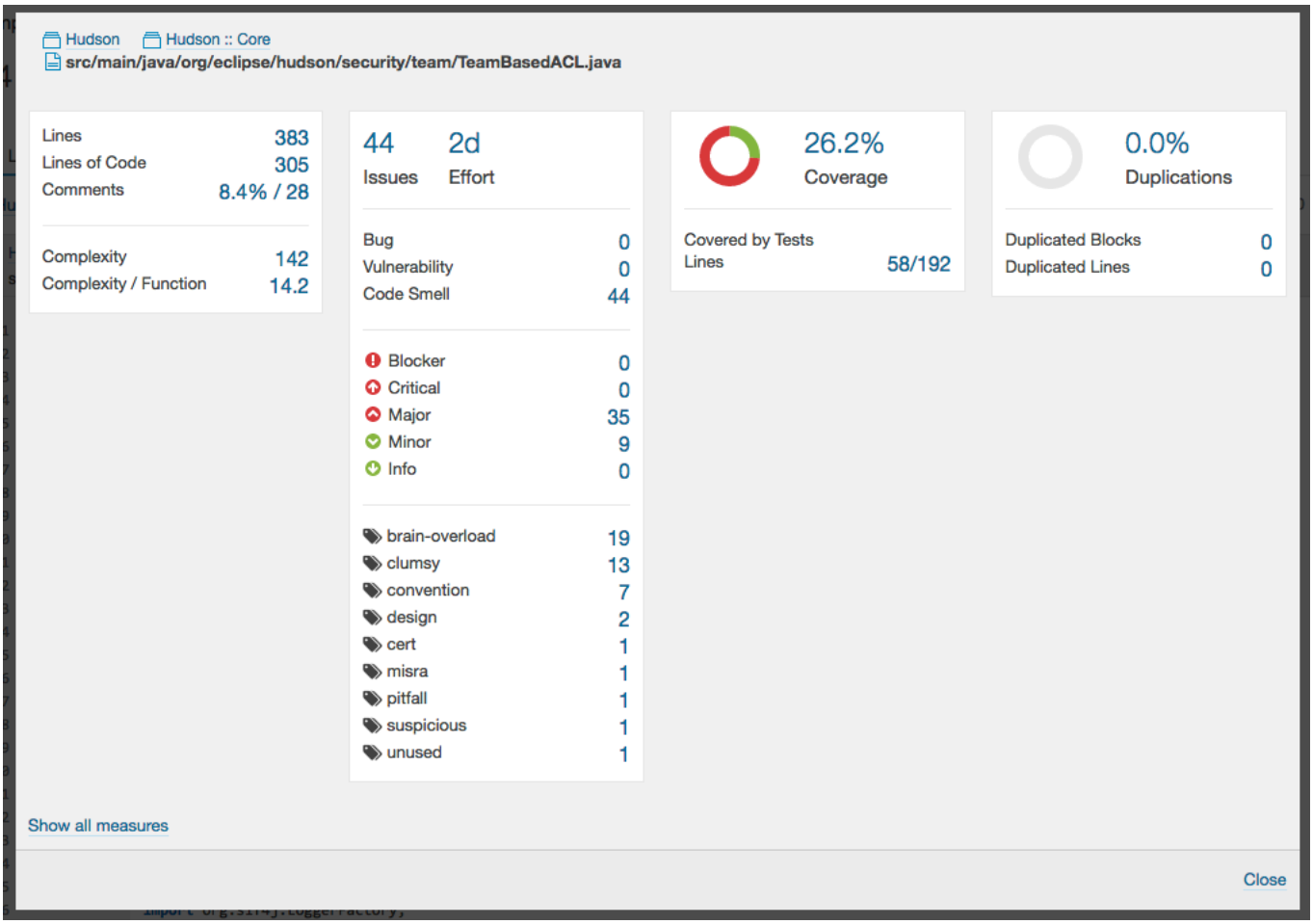
Un clic sur le nom du fichier montre :

- le code annoté : barres verticales verte pour le code couvert, rouge non couvert, orange partiellement couvert, icônes de niveau de sévérité marquant les *issues* détectés dans le code, nom de l'utilisateur qui a fait le commit devant la ligne :

- des mesures pour le fichier : nombre de lignes, nombre *d'issues*, taux de couverture.

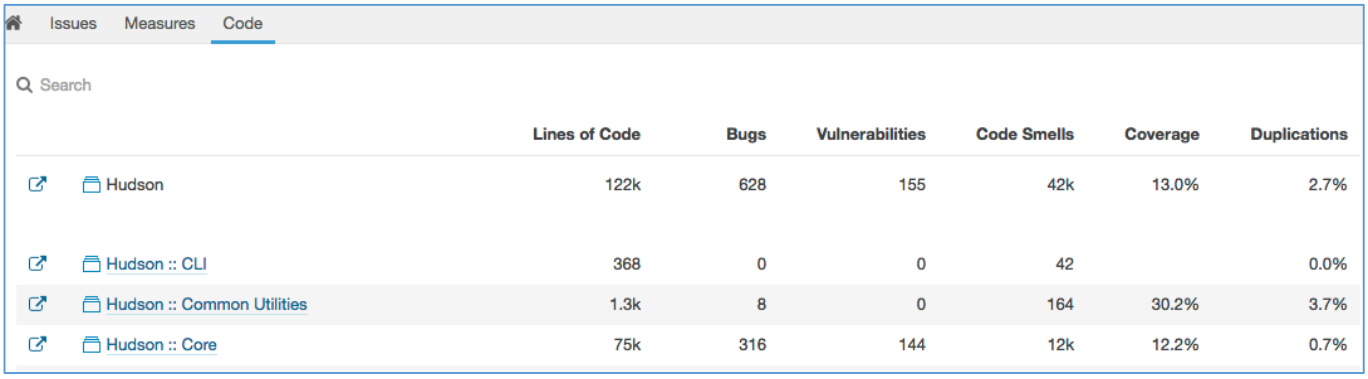


Un détail des mesures pour le fichier peut être obtenu à partir du menu *Show Measures* :



11. Code

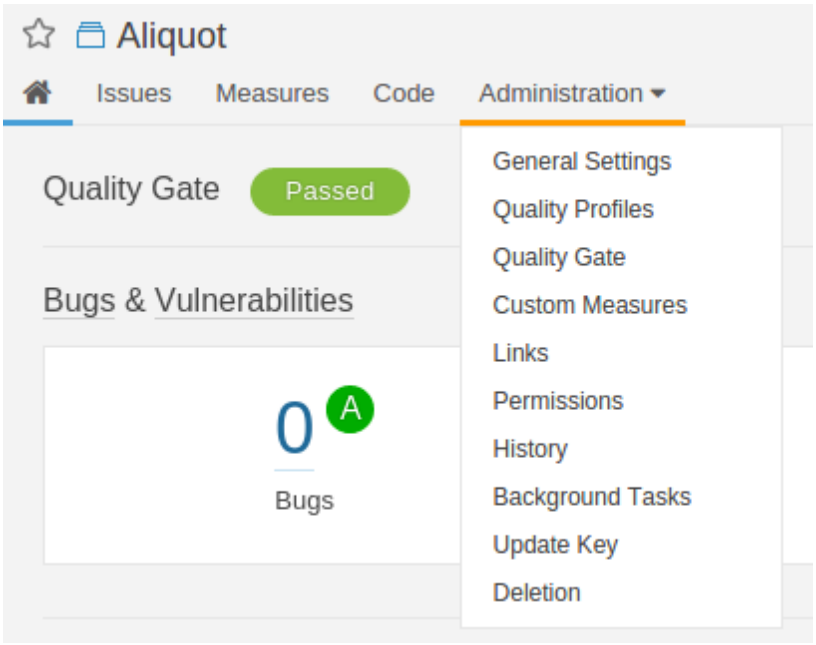
Le menu *Code* donne des informations pour chaque paquet : lignes de code, nombre de bugs, de vulnérabilités, de code smells, taux de couverture, duplications :

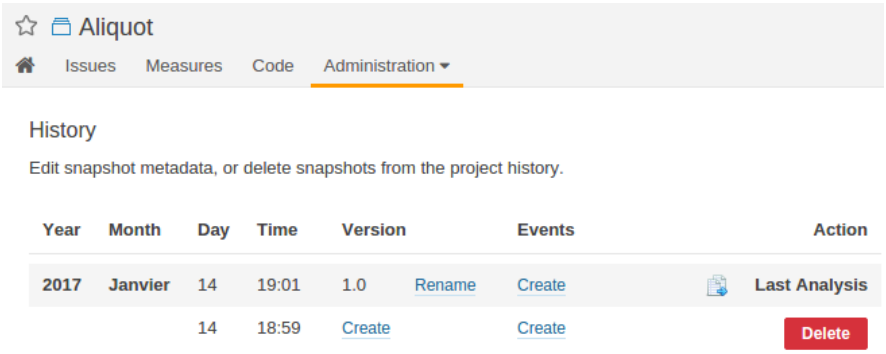


	Lines of Code	Bugs	Vulnerabilities	Code Smells	Coverage	Duplications
Hudson	122k	628	155	42k	13.0%	2.7%
Hudson :: CLI	368	0	0	42		0.0%
Hudson :: Common Utilities	1.3k	8	0	164	30.2%	3.7%
Hudson :: Core	75k	316	144	12k	12.2%	0.7%

12. Supprimer des analyses

Administration > *History* dans le menu d'un projet



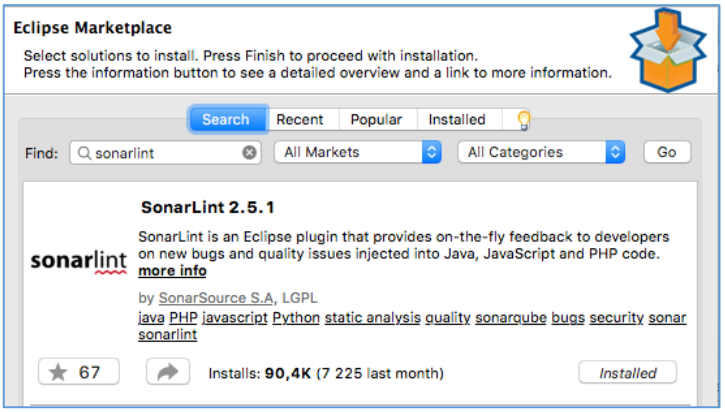


7) Inspection en continu du code avec SonarLint

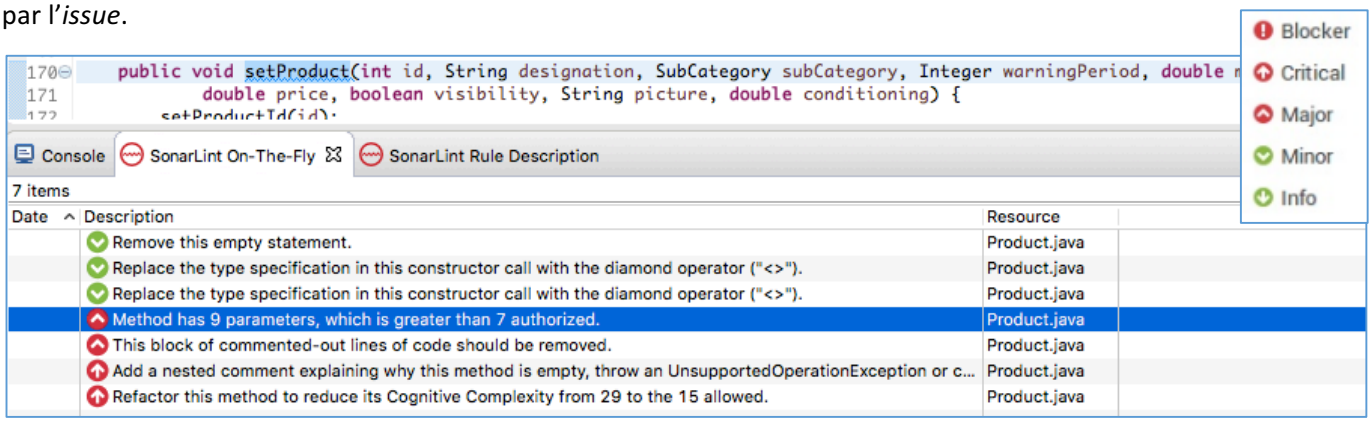
Le code source Java, PHP, JavaScript et Python peut être analysé avant le commit directement dans l'IDE Eclipse avec le plugin SonarLint :

http://www.sonarlint.org/eclipse/

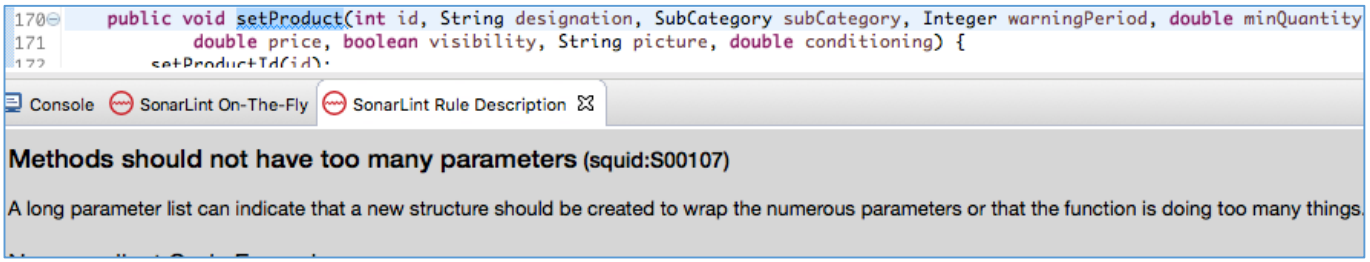
La version testée est la 2.5.1 du 12 janvier 2017.



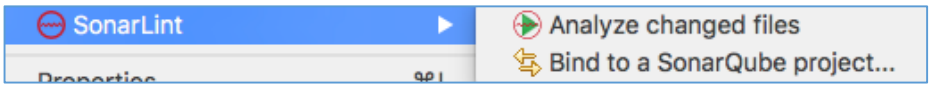
La vue *Sonar-Lint On-The-Fly,* dans l'onglet sous le code, affiche la liste des *issues* avec une icône indiquant leur niveau de sévérité. La vue est disponible à partir du menu *Window > Show View > Other…* Un clic sur l'*issue* montre le code concerné par l'*issue*.

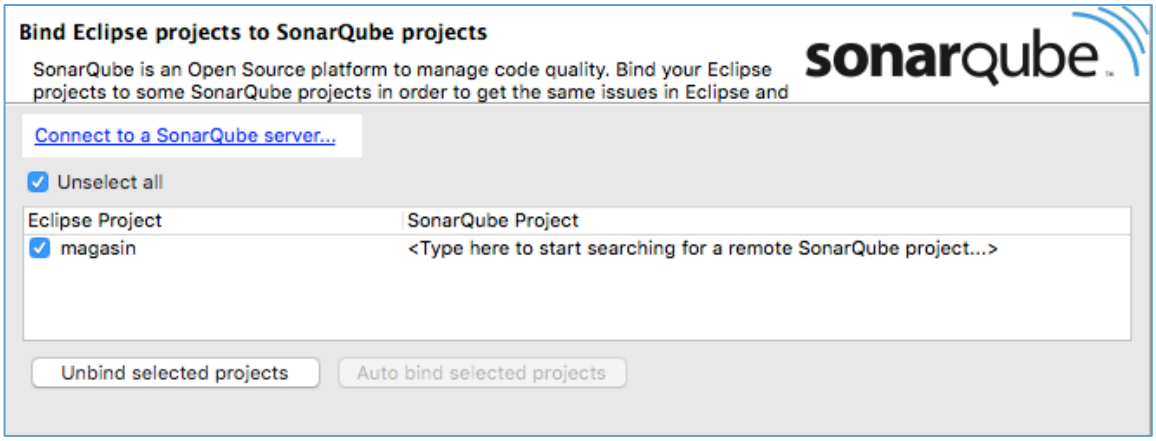


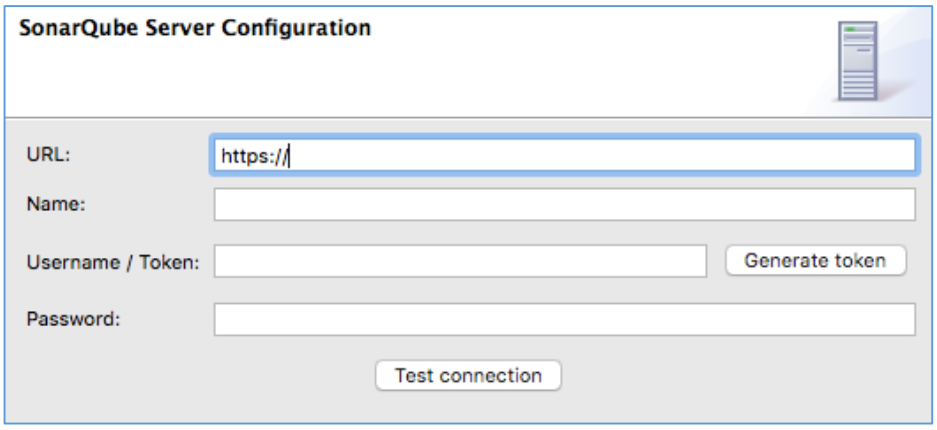
La vue *SonarLint Rule Description* affiche la description de la règle pour laquelle un problème a été détecté.



Le plugin peut fonctionner sans serveur ou en **mode connecté**, ce qui permet d'utiliser le profil qualité et les réglages définis pour le projet sur le serveur SonarQube. Se placer sur le répertoire du projet et le lier à un projet d'un serveur SonarQube (clic droit + *SonarLint > Bind to a SonarQube project…*).







Un plugin SonarLint est également disponible pour IntelliJ (PHP, Java, JavaScript)

https://plugins.jetbrains.com/idea/plugin/7973-sonarlint

Références

- Documentation : http://docs.sonarqube.org/display/SONAR/Documentation
- Analyser le code : http://docs.sonarqube.org/display/SONAR/Analyzing+with+SonarQube+Scanner
- Fonctionnalités : http://www.sonarqube.org/features/
- Liste des plugins : https://docs.sonarqube.org/display/PLUG/Plugin+Library
- Leak : https://docs.sonarqube.org/display/HOME/Fixing+the+Water+Leak
- Liste des utilisateurs SonarQube : https://www.sonarsource.com/customers/
- Inspection continue : « Continuous Inspection – a paradigm shift in software quality management » https://www.sonarsource.com/resources/white-papers/continuous-inspection.html