



# Mise en œuvre de la qualité avec SonarQube

---

David THIBAU – 2021

[david.thibau@gmail.com](mailto:david.thibau@gmail.com)



# Agenda

---

## **Introduction à la qualité**

- Analyse continue, Métriques logiciels, modèle qualité, Charte qualité

## **SonarQube**

- L'offre, Architecture, Installation, Concepts, Métriques, Workflow

## **Mise en place**

- Recommandations générales, Configuration de l'analyse, Personnalisation profils et portes qualité, Règles personnalisées

## **SonarLint**

- Installation IDE

## **Administration et intégration**

- Sécurité, Historique, WebHooks, Intégration Jenkins, Exploitation



# Introduction à la qualité

---

## **Introduction** Métriques qualité



# Constat

---

En moyenne 80% du coût d'un logiciel concerne la maintenance

Le coût de maintenance est très variable en fonction de la **qualité interne** du software.

=> Le niveau de maintenabilité d'un logiciel déterminera son coût financier global



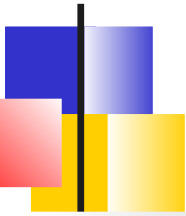
# Approche standard

---

Les approches traditionnelles du contrôle de la qualité des codes impliquaient des audits ponctuels généralement effectués par des auditeurs externes.

Cela pouvait entraîner des perturbations dans le cycle de développement : retards, retouches, remises en cause des choix

=> Tout cela avait tendance à donner une image coûteuse de la qualité .



# Hétérogénéité des exigences

---

Les approches traditionnelles qui évaluent la qualité en se basant sur des valeurs absolues, telles que le nombre total de problèmes trouvés lors d'un audit, ne sont pas viables car elles sous-entendent que tous les projets ont les mêmes objectifs de qualité

- un projet existant peut ne pas être tenu au même niveau de qualité qu'un nouveau projet
- Le développement interne pourra être jugé différemment du code externalisé.

=> Obliger chaque projet à atteindre les mêmes valeurs absolues avant la mise en production est donc souvent impraticable.



# Analyse continue

---

L'**analyse continue** consiste à intégrer la démarche qualité dans le cycle de développement du logiciel

Le concept-clé est de trouver les problèmes le plus tôt possible ; lorsqu'il est facile et peu coûteux de les corriger.

Des audits automatisés sont effectués quotidiennement et les résultats sont mis à disposition des acteurs du projet.

Cela a d'énormes effets sur le ROI d'un projet de développement



# Amélioration continue

---

Des audits peuvent être également effectués continuellement dans l'environnement du développeur.

- lorsque le code est encore présent à l'esprit et qu'il n'est pas encore enregistré dans le dépôt de source.

Cela permet au développeur de s'améliorer et de se débarrasser de ses mauvaises pratiques de codage.

L'aspect collaboratif de l'analyse continue permet également une appropriation collective du code et des bonnes pratiques.





# Les 10 principes (1)

---

1. Tous les prenant-part au processus de développement (pas seulement les développeurs) ont un accès facile aux données significatives.
2. La gestion de la qualité concerne tout le monde dès le début de projet. L'équipe de développement en a la responsabilité ultime.
3. Les exigences qualité sont des éléments de recette du logiciel.
4. Elles doivent être objectives
5. Le plus possible, elles doivent être communes à tous les projets de développement.



# Les 10 principes (2)

---

6. La qualité du logiciel doit être à jour et concerner la dernière version du code.
7. Les logiciels doivent être analysés en permanence afin de corriger les problèmes au + tôt.
8. Les parties prenantes doivent être alertés lorsque de nouveaux défauts sont détectés
9. Les données de qualités doivent être disponibles en valeur absolue et valeur différentielle (nouveau code seulement)
10. La résolution des défauts doit être clairement planifiée et provisionnée.



# Introduction à la qualité

---

## Introduction **Métriques qualité**



# Métriques

---

La mesure est fondamentale à toutes les disciplines d'ingénierie ;  
le software n'est pas une exception

Dans le cadre d'un logiciel, la production de métriques permet :

- Donner un chiffre à des facteurs qualitatifs
- De s'améliorer : améliorer le processus de développement, la qualité du produit
- D'affiner les estimations (prix, délais)

Les métriques sont de 2 types :

- Interne : analyse sur le code source ou compilé (qualité, couverture de test, ...)
- Externe : Sur le livrable (Performance, Disponibilité de service, Nbre d'erreurs ...)



# Norme ISO-9126

---

La norme ISO-9126 est la première norme établissant un modèle qualité de référence

- Elle normalise un nombre important de critères qualité
- Et surtout de métriques (façon de mesurer)

Elle a ensuite été complétée par la norme 25010 et par une série de standard dénommée **SQuaRE** (*Software product Quality Requirements and Evaluation*)



# ISO 25010

## SOFTWARE PRODUCT QUALITY

### Functional Suitability

- Functional Completeness
- Functional Correctness
- Functional Appropriateness

[iso25000.com](http://iso25000.com)

### Performance Efficiency

- Time Behaviour
- Resource Utilization
- Capacity

### Compatibility

- Co-existence
- Interoperability

### Usability

- Appropriateness
- Recognizability
- Learnability
- Operability
- User Error Protection
- User Interface Aesthetics
- Accessibility

### Reliability

- Maturity
- Availability
- Fault Tolerance
- Recoverability

### Security

- Confidentiality
- Integrity
- Non-repudiation
- Authenticity
- Accountability

### Maintainability

- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

### Portability

- Adaptability
- Installability
- Replaceability



# Définitions (1)

---

***Functionality (Fonctionnalité)*** : Capacité à délivrer les fonctionnalités attendues

***Reliability (Fiabilité)*** : Capacité à maintenir le niveau de performance attendu dans les conditions d'utilisation spécifiées

***Usability (Utilisabilité)*** : Capacité à être compris, appris, utilisé, convivial pour l'utilisateur dans les conditions d'utilisation spécifiées



# Définitions (2)

---

***Efficiency (Efficacité)*** : Capacité à fournir les performances appropriées selon les ressources utilisées dans les conditions spécifiées

***Maintainability (Maintenabilité)*** : Capacité à être modifié (corrigé, amélioré, adapté) selon les changements  
(environnement/spécification)

***Portability (Portabilité)*** : Capacité à être transféré d'un environnement à un autre





# Sécurité

**Security (Sécurité)** fait également partie de la norme qualité. Elle définit les sous-caractéristiques suivantes :

- Confidentialité : Données accessibles par les seules personnes autorisées.
- Intégrité : Protection contre la modification de données
- Non-répudiation : Actions prouvées
- Traçabilité : Action pouvant être tracées
- Authenticité : Identité prouvée



# Charte qualité

---

La norme ne donne que les métriques

- Elle ne donne pas d'indicateurs
- Exemple : pour l'adéquation fonctionnelle
  - La norme dit « plus c'est proche de 1, mieux c'est »
  - A combien est-ce suffisant ?

La charte qualité (profil qualité Sonar)

- Permet d'appliquer des seuils et des critères à des métriques pour obtenir des indicateurs



# Charte qualité

---

Une charte qualité est constituée

- D'un modèle qualimétrique
  - Il indique ce que l'on mesure et comment on le mesure
- D'un ensemble d'exigences
  - Seuils désirables ou contractuels

La comparaison des valeurs mesurées aux exigences produira des indicateurs

- Acceptation ou rejet d'une livraison
- Estimation de la probabilité de défauts
- Estimation des coûts de maintenance
- Etc...



# Indicateurs

---

Le modèle qualimétrique est neutre

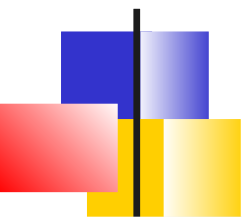
- « formule de calcul du nombre de lignes de code »

La valeur de mesure est neutre

- « nombre de lignes de code=100 »

Le jugement est porté dans une charte qualité

- Spécifique, en fonction des exigences qualité définies
  - Charte qualité Java :  
nombre de lignes de code d'une méthode  $> 20 \rightarrow$  problème sérieux
  - Charte qualité COBOL :  
nombre de lignes de code d'une COPY  $> 200 \rightarrow$  problème sérieux



# SonarQube

---

## **Offre et architecture**

Installation

Concepts Sonar

Calculs de métriques

Administration utilisateurs

Workflow



# SonarQube

---

**SonarQube** est devenu l'outil standard de facto qui regroupe tous les outils de calcul de métrique interne d'un logiciel (toute technologie confondue)

Il intègre 2 aspects :

- Définitions des règles de codage du projet  
=> Détection des transgressions et estimation de la dette technique
- Calculs des métriques internes
- Définition de porte qualité en fonction des métriques calculés

Sa mise en place nécessite une adaptation en fonction du projet.



# Offre Sonar (1)

---

Modèle Open Source : Versions communautaire et payantes (Developer et enterprise et DatCenter)

- <http://www.sonarqube.org/> : développement / support

Analyse complète de la qualité d'une application

- Nombreux métriques
  - Quantitatifs : nombre de classes, duplication de code, ...
  - Qualitatifs : couverture et taux de réussite des tests, complexité du code, respect des règles de codage...

Tableau de bord projet

Historique des statistiques

Gestion des règles de codage, mise à jour

Gestion de profils et portes qualité

Workflow autour de la résolution des issues

Visualisation du code source, accès aux dépôts de sources

Intégration CI/CD



# Offre Sonar (2)

---

Utilisation de nombreux outils via des plugins

- CheckStyle
- PMD
- FindBugs
- Cobertura, Jacoco
- Sonar Squid
- ...

Gestion des plugins via la MarketPlace : Certains nécessitent l'acquisition de licences





# Edition Developer

---

- C, C++, Obj-C, Swift, ABAP, T-SQL, PL/SQL support
- Intégration SCM : Analyse des branches (longues et courtes) et pull requests (analyse du new code)
- Décoration pull-request
- Détection de défaut d'injection Java, C#, PHP, Python, Javascript, Typescript



# Edition entreprise

---

- Gestion de Portfolio (suivi transverse de plusieurs projets)
- Reporting PDF, reporting sécurité
- Transfert de projet entre instances Sonar
- Traitement // des rapports d'analyses
- Support pour Apex, COBOL, PL/I, RPG, VB6
- Support technique

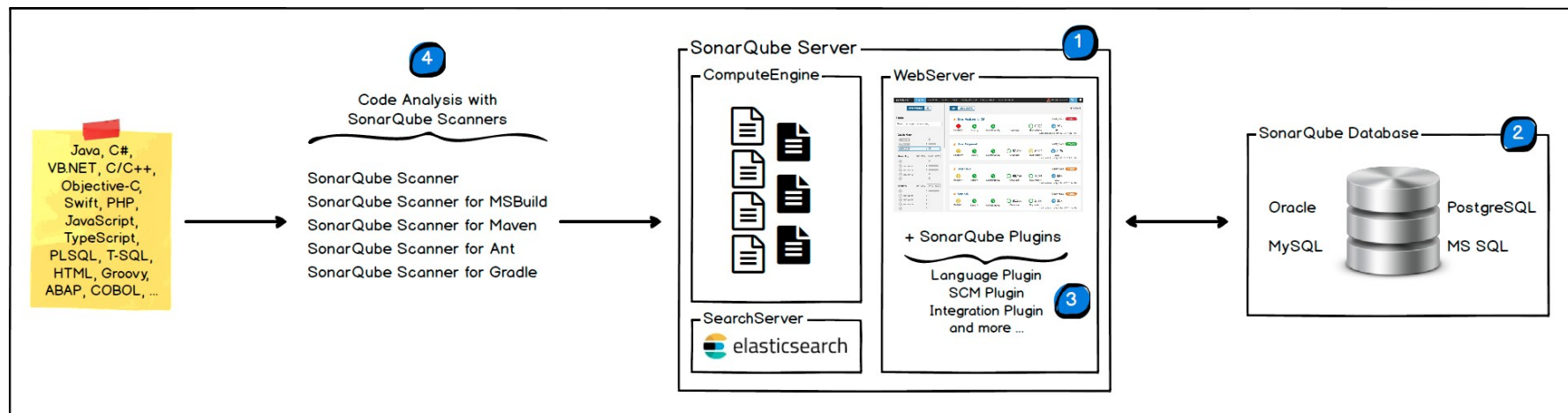


# Data Center

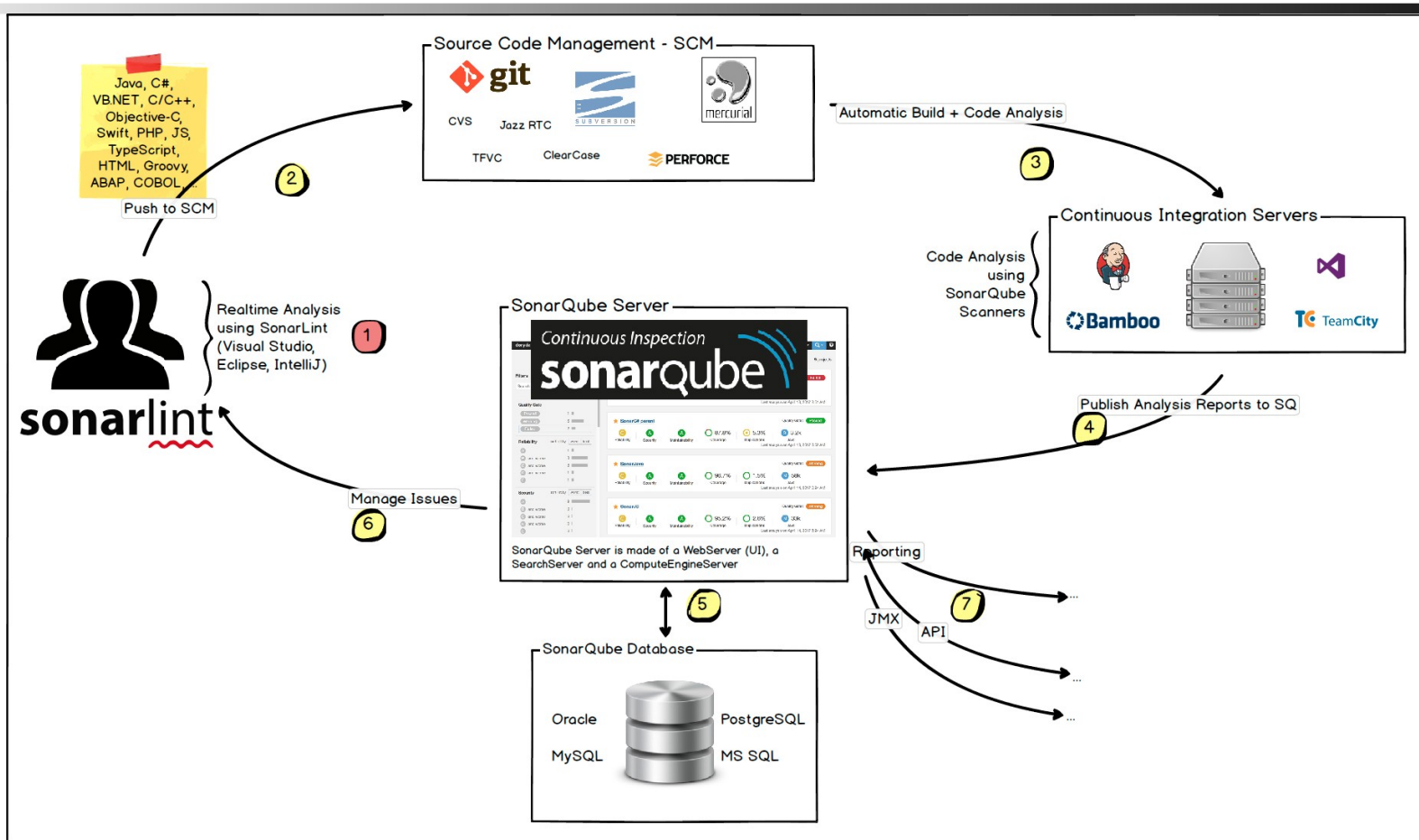
---

- Redondance des composants
- Résilience des données
- Scalabilité horizontale

# Architecture



# Analyse continue





# SonarQube

---

Offre et architecture

**Installation**

Concepts Sonar

Calculs de métriques

Administration utilisateurs

Workflow



# Releases

---

Le développement de Sonar est en mode agile, donc des releases sont fréquemment effectuées.

Certaines releases sont marquées comme **LTS** (Long Term Support)

- Elles sont + stables en terme de fonctionnalités ~ 1an
- Sonar Source s'engage à corriger les bugs
- Lors d'une mise à jour, il y a plus de travail à effectuer



# Hardware

---

Sonar nécessite :

- 2 Go de RAM
- Espace disque en fonction du nombre de projets.  
*SonarCloud* avec plus de 300 projets utilise 15Go
- Disques rapides de préférence





# Pré-requis

---

Pour l'installation SonarQube nécessite une JRE

- Version 8.X et 9.x : Java 11
- Version 6.x : Java 1.8

Une base de données :

- BD embarquée pour évaluation
- PostgreSQL, MSQl, Oracle

ElasticSearch

- Pour Linux :
  - *vm.max\_map\_count* > 262144
  - *fs.file-max* > 65536
  - L'utilisateur de SonarQube peut ouvrir 65536 fichiers
  - Et démarrer 2048 threads



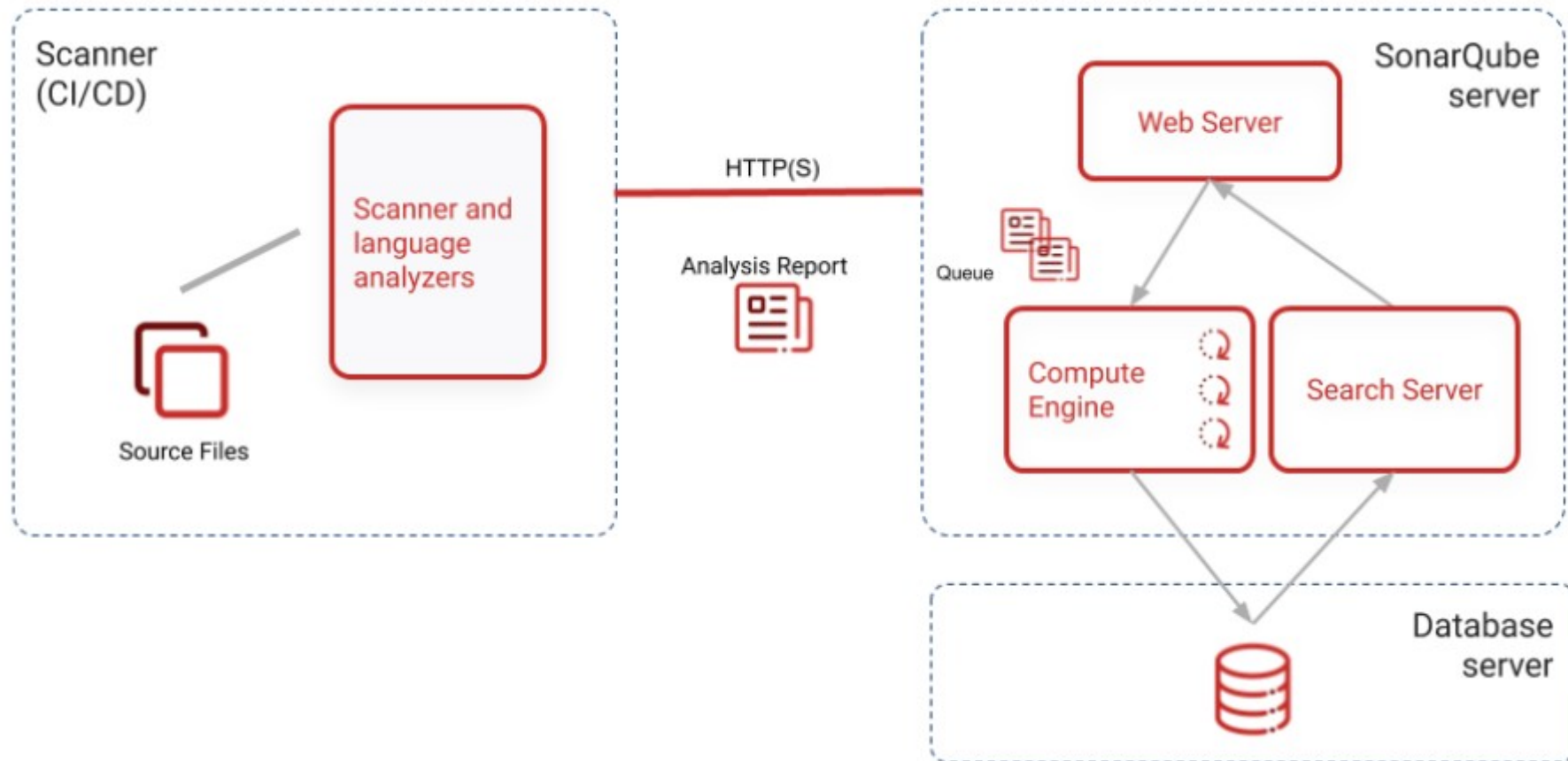
# Distribution

---

La distribution de sonar est disponible sous forme d'archive zip, il suffit de la décompresser

Dans un environnement Linux, des packages sont également disponibles

# Composants SonarQube





# Rôle des composants

---

Le serveur SonarQube exécute 3 processus :

- Le serveur Web qui sert l'interface utilisateur de SonarQube.
- Le serveur de recherche basé sur Elasticsearch.
- Le moteur de calcul en charge de traiter les rapports d'analyse de code et de les enregistrer dans la base de données SonarQube.

La base de données stocke les éléments suivants :

- Métriques et problèmes de qualité et de sécurité du code générés lors des analyses de code.
- La configuration de l'instance SonarQube.

Un ou plusieurs scanners exécutés sur les serveurs de build pour analyser les projets envoient leur rapports vers le serveur.



# Base de données

---

SonarQube est livré avec une base de données embarquée, cela n'est pas recommandé en production

Pour installer une autre base :

- Créer un schéma vide et un utilisateur *sonarqube*
- Donner à l'utilisateur *sonarqube*, les droits de créer, mettre à jour et supprimer des tables
- Configurer l'accès JDBC
- Pour une base Oracle, ajouter le driver JDBC



# Configuration JDBC

---

Dans le fichier ***conf/sonar.properties***

```
sonar.jdbc.username=postgres  
sonar.jdbc.password=postgres  
sonar.jdbc.url=jdbc:postgresql://localhost/sonar
```

... éventuellement, configuration du pool de connexions



# ElasticSearch

---

Par défaut, ElasticSearch stocke ses index dans *\$SONARQUBE-HOME/data*

- Cela n'est pas recommandé en production.

Idéalement choisir un volume dédié avec des E/S rapides.

- Cela facilite la mise à niveau de SonarQube et améliore les performances

```
sonar.path.data=/var/sonarqube/data
```

```
sonar.path.temp=/var/sonarqube/temp
```



# Serveur web

---

Le port par défaut est 9000 avec un chemin de contexte à /

Ces valeurs peuvent être modifiées dans *conf/sonar.properties*

```
sonar.web.host=192.0.0.1
```

```
sonar.web.port=80
```

```
sonar.web.context=/sonar
```





# Choisir la bonne version de Java

---

Si il existe plusieurs versions de Java sur le serveur cible.

La version utilisée peut être précisée dans

`$SONARQUBE-HOME/conf/wrapper.conf`

Et la ligne

`wrapper.java.command=/path/to/my/jdk/bin/java`



# Démarrage

---

Linux/Mac OS :

`bin/<YOUR OS>/sonar.sh start`

Windows :

`bin/windows-x86-XX/StartSonar.bat`

Fichier de traces : `logs/sonar.log`

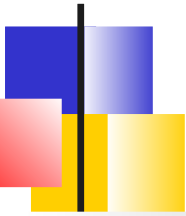
Un utilisateur *admin/admin* est initialement créé

Les options JVM de démarrage peuvent également être indiquées dans  
*conf/sonar.properties*

`sonar.web.javaOpts=-server`

On peut obtenir les traces en mode DEBUG avec :

`sonar.log.level=DEBUG`



# Installation service Windows

---

SonarQube fournit des scripts  
d'installation de service Windows :

```
%SONARQUBE_HOME%/bin/windows-x86-32/InstallNTService.bat  
%SONARQUBE_HOME%/bin/windows-x86-32/  
UninstallNTService.bat
```

Le service peut ensuite être  
démarré/stoppé par :

```
%SONARQUBE_HOME%/bin/windows-x86-32/StartNTService.bat  
%SONARQUBE_HOME%/bin/windows-x86-32/StopNTService.bat
```



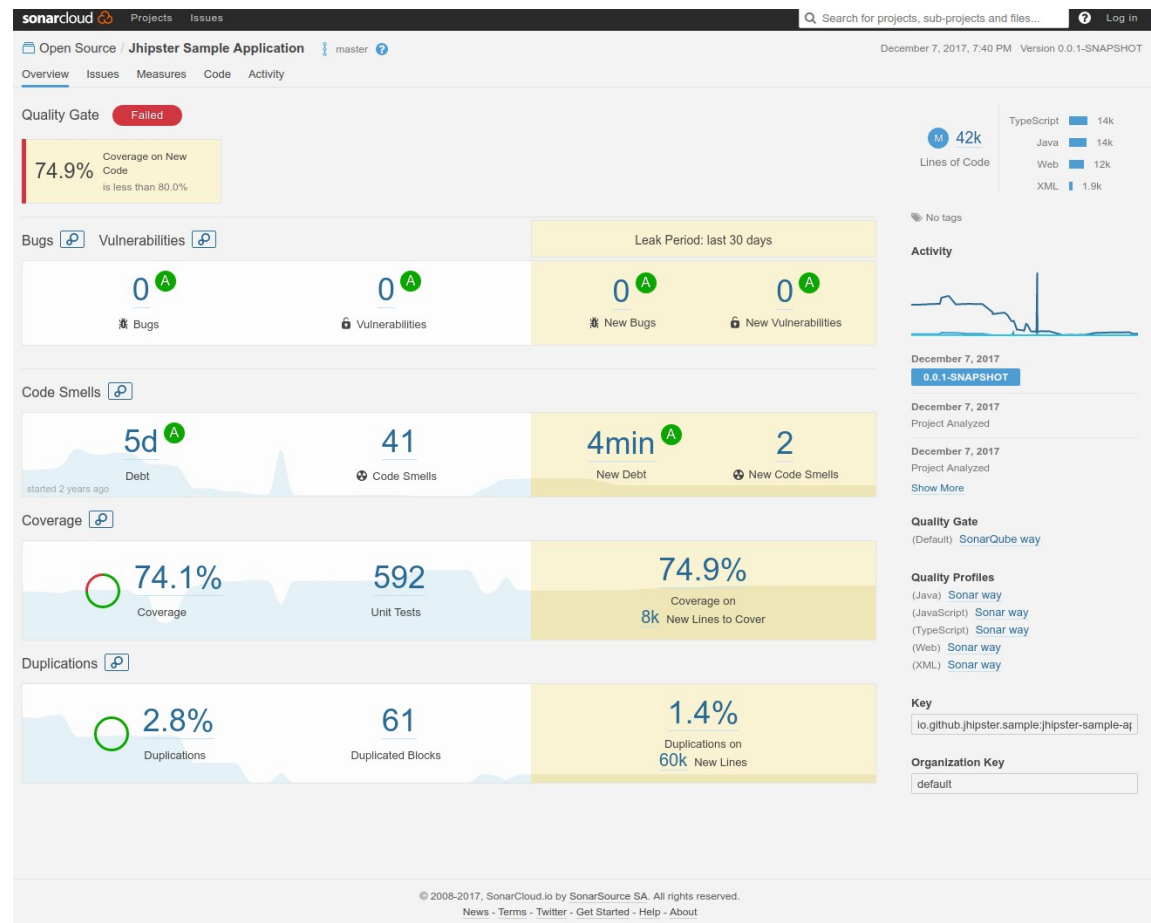
# Installation Service Linux

---

Voir :

<https://docs.sonarqube.org/display/SONAR/Running+SonarQube+as+a+Service+on+Linux>

# Sonar Dashboard





# SonarQube

---

Offre et architecture  
Installation

**Concepts Sonar**

Calculs de métriques  
Administration utilisateurs  
Workflow



# Analyse et Scanners

---

*Sonar* permet de démarrer une analyse facilement grâce aux **Scanners** fournis :

- MSBuild, Maven, Gradle, Ant, Jenkins, Azure DevOps Commande en ligne

Ceux-ci prennent généralement en paramètre un fichier de configuration de l'analyse qui est propre au projet

Le projet SonarQube est créé :

- Soit à l'avance
- Soit lors de la première analyse



# Analyse et langages

---

*SonarQube* supporte plus de 20 langages

Cependant, ce qui est analysé dépend du langage :

- Pour tous les langages, le code source peut être importé d'un SCM. (ainsi que les données « blame »)  
Git et SVN sont supportés, les autres nécessitent des plugins.
- Pour tous les langages, une analyse statique du code source est faite
- Pour certains langages, le code compilé est également analysé statiquement (.class Java, .dll C#, etc.)
- Une analyse dynamique peut être effectuée pour certains langages.
- Tous les fichiers reconnus par l'édition de *SonarQube* sont automatiquement analysés





# Étapes d'une analyse

---

L'analyse s'effectue en plusieurs étapes :

1. Des données sont demandées au serveur par le scanner
2. Les fichiers soumis à l'analyse sont analysés et les données résultantes sont renvoyées au serveur sous la forme d'un rapport
3. Le rapport est ensuite analysé de manière asynchrone côté serveur.  
Les rapports sont mis en file d'attente et traités séquentiellement. Un statut des tâches est disponible dans l'interface



# Analyse

---

## Base de règles

- Les règles sont fournies par SonarSource ou par des plugins
- On peut rajouter ses propres règles
- Elles peuvent être activées/désactivées dans des « profils qualité »

L'analyse consiste à vérifier les règles du *profil qualité* associé au projet.

Lorsqu'une règle est transgressée, elle produit une **Issue**

- *Bugs* : Incidence sur la fiabilité du logiciel
- *Code smells* : Incidence sur la maintenabilité
- *Vulnérabilité* : Incidence sur la sécurité






# Règle

---

## "hashCode" and "toString" should not be called on array instances

squid:S2116  

 Bug  Major  No tags Available Since December 8, 2017 SonarAnalyzer (Java) Constant/issue: 5min

While `hashCode` and `toString` are available on arrays, they are largely useless. `hashCode` returns the array's "identity hash code", and `toString` returns nearly the same value. Neither method's output actually reflects the array's contents. Instead, you should pass the array to the relevant static `Arrays` method.

### Noncompliant Code Example

---

```
public static void main( String[] args )
{
    String argStr = args.toString(); // Noncompliant
    int argHash = args.hashCode(); // Noncompliant
}
```

---



# Profil Qualité

---

Les ***profils qualité*** définissent les règles pour un projet particulier

*SonarSource* fournit un profil qualité par défaut pour chaque langage

Profil Java : 292 règles

- 93 bugs
- 18 vulnérabilités
- 181 Code smells



# Métriques

---

Sonar définit plusieurs axes pour les métriques qualité associés à un projet :

- Complexité : Formule
- Documentation : Formule
- Duplications : Nombre de Copié/Collé
- Fiabilité : Nombre de bugs
- Maintenabilité : Nombre de code smells
- Sécurité : Nombre de vulnérabilités
- Tests : Couverture des tests



# SonarQube - Progression

---

SonarQube propose 2 moyens pour évaluer les progrès depuis le démarrage d'une démarche qualité :

- **New Code** :  
Permet de définir un point d'origine. *Sonar* montre alors des métriques par rapport au code introduit depuis ce point d'origine. (Typiquement release précédente ou début du Sprint)
- Les **portes qualité**  
Permet de définir un ensemble de seuils pour différentes mesures. Cela répond à la question : « Le code peut il aller en production ? » Les seuils progressent au fur et à mesure des releases.



# Porte qualité

---

Les portes qualité sont différentes en fonction des projets

La porte qualité « *SonarQube way* » est fournie par *SonarSource* et elle est activée par défaut

Chaque condition d'une porte qualité est la combinaison de :

- La métrique concerné
- La période : Date de la mesure ou différentiel par rapport au point d'origine (***Leak period***)
- Des seuils pour un avertissement ou pour une erreur



# SonarQube

---

Offre et architecture

Installation

Concepts Sonar

**Calculs de métriques**

Administration utilisateurs

Workflow





# Complexité

---

## **Complexité cyclomatique :**

- Calculé à partir du nombre de chemins possibles dans le code. Chaque fonction a une complexité minimale de 1.

## **Complexité Cognitive :**

- Difficulté à comprendre les chemins dans le code  
*Voir <https://www.sonarsource.com/resources/white-papers/cognitive-complexity.html>*

## **Mesures dérivées :**

- Complexité moyenne par classe.
- Complexité moyenne par fichier
- Complexité moyenne par méthode.



# Documentation

---

## **Commentaires :**

- Nombre de lignes comportant une ligne de commentaire ou du code commenté.
- Nombre de lignes de code commentées
- Pourcentage de lignes de commentaires par rapport au total de lignes

## **Méthodes publiques et API**

- Pourcentage de l'API publique documentée :
- Nombre de méthodes publiques nom commentées.



# Duplications

---

## **Duplications**

- Nombre de blocs de lignes dupliqués
- Nombre de fichiers impliqués dans les duplications.
- Nombre de lignes impliqués dans les duplications.
- Pourcentage de lignes dupliquées



# Fiabilité Bugs

---

## **Bugs , New Bugs :**

- Nombre de bugs et de nouveaux bugs, par sévérité, par statuts

## **Indicateur de fiabilité** (*Reliability rating*)

- A = 0 Bug
- B = au moins 1 bug mineur
- C = au moins 1 bug majeur
- D = Au moins 1 critique
- E = Au moins 1 bloquant

## **Effort de correction** (*Reliability remediation effort*) :

- L'effort pour fixer tous les bugs ou tous les nouveaux bugs (en minutes et en se basant sur une journée de 8 heures),



# Sécurité

---

## **Vulnérabilités :**

- Nombre de vulnérabilités et de nouvelles vulnérabilités, par sévérité, par statut

## **Indicateur de sécurité** (*Security Rating*)

- A = Aucune vulnérabilité
- B = au moins 1 vulnérabilité mineure
- C = au moins 1 vulnérabilité majeure
- D = au moins 1 vulnérabilité critique
- E = au moins 1 vulnérabilité bloquante

## **Effort de sécurisation** (*Security remediation effort*) :

- Minutes pour corriger les vulnérabilités, sur tout le code, sur le nouveau code



# Security Hotspot

---

Un ***Security Hotspot*** met en évidence un morceau de code sensible à la sécurité que le développeur doit examiner.

Après une revue, on décide si la menace est réel et éventuellement on applique un correctif.

Un statut est associé au security hotspot :

- *To review*
- *reviewed*



# Maintenabilité

---

## **Code Smells**

- Nombre de code smells, de nouveaux par sévérité, par statut

## **Dettes techniques :**

- Effort nécessaire pour corriger tous les problèmes de maintenabilité (en minutes).

## **Rapport dette/développement :**

- Rapport entre le coût de développement et le coût pour fixer les problèmes de maintenance (Le coût d'une ligne de développement est estimé à 0,06 jour)

## **Taux de maintenabilité :** Indicateur en fonction du rapport précédent

- $A < 0,05$  /  $B < 0,1$  /  $C < 0,2$  /  $D < 0,5$  /  $E > 0,5$



# Couverture des tests

---

Il s'agit de vérifier le code utile exécuté durant l'exécution des classes de tests.  
2 axes de vérification :

- Les lignes
- Les chemins ou conditions

Java : Sonar s'appuyait vers le projet OpenSource Cobertura; dans les dernières versions il supporte ***jacoco***.





# Métriques

---

## **Couverture des lignes :**

- Lignes à couvrir
- Lignes couvertes
- Pourcentage

## **Couverture des conditions :**

- Conditions à couvrir
- Conditions couvertes
- Pourcentage



# Mesures globales

---

**Couverture** : Mélange entre la couverture de lignes et la couverture de conditions

– **Couverture =  $(CT + CF + LC)/(2*B + EL)$**

- CT = Conditions ayant été évaluées à 'true' au moins une fois
- CF = Conditions ayant été évaluées à 'false' au moins une fois
- LC = Lignes à couvrir - lignes non couvertes
- B = Nombre total de conditions
- EL = Nombre total de lignes à couvrir

## Couverture des conditions

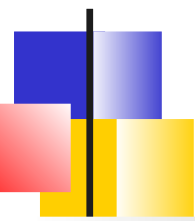
– **Couverture de condition =  $(CT + CF) / (2*B)$**

- CT = Conditions ayant été évaluées à 'true' au moins 1 fois
- CF = Conditions ayant été évaluées à 'false' au moins 1 fois
- B = Nombre Total de conditions

## Couverture des lignes

– **Taux de couverture de ligne =  $LC / EL$**

- LC = Lignes à couvrir moins les lignes non-couvertes
- EL = Lignes à couvrir



# Rapport d'exécution des tests

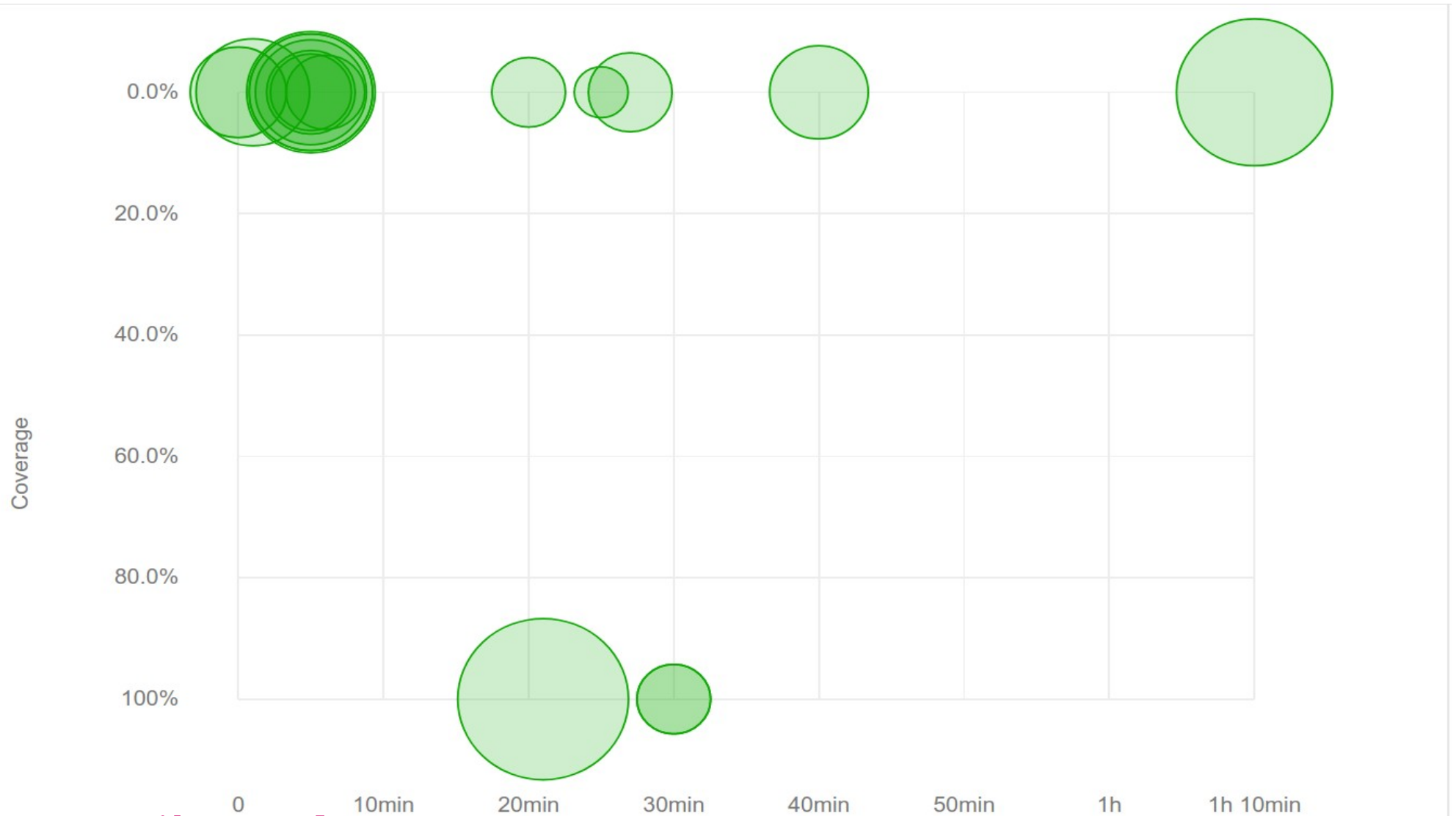
---

Sonar remonte également un rapport sur l'exécution des tests unitaires :

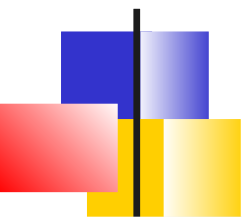
- Nombre de tests unitaires
- Nombre de tests unitaires ignorés
- Nombre de tests ayant échoués (Erreur d'assertion)
- Nombre de tests en erreur (Exception).
- Densité de réussite
- Durée des tests
- Nombre de conditions non couvertes par les tests
- Lignes non couvertes

# Indicateurs Visuels

## Sonar Bulles



*TP 2 : Première analyse*



# SonarQube

---

Offre et architecture

Installation

Concepts Sonar

Calculs de métriques

**Administration utilisateurs**

Workflow



# Administration

---

Le menu administration propose plusieurs entrées :

- Configuration générale ou par technologie :  
Nettoyage de la base, Serveur SMTP, leak\_period, CI Serveur, SCM,  
Métriques personnalisés.  
Certaines configuration peuvent être surchargées au niveau projet
- Utilisateurs, groupes, permissions
- Gestion des projets : suppression, droits
- Surveillance du système
- Marketplace et gestion des plugins



# Sécurité

---

SonarQube intègre un mécanisme complet pour gérer l'authentification, l'autorisation et le cryptage des mots de passe

Il est cependant possible d'externaliser la sécurité vers :

- LDAP ou Active Directory - SonarQube LDAP Plugin
- PAM - SonarQube PAM Plugin
- Crowd - SonarQube Crowd Plugin
- GitHub - GitHub Authentication Plugin
- Bitbucket - Bitbucket Authentication Plugin

Les utilisateurs de la base sont alors soit locaux soit externes



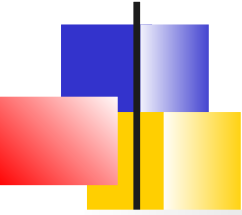
# Démarrage

---

Par défaut, une authentification est nécessaire pour accéder au résultats

Au démarrage, il y a un seul compte administrateur (admin/admin) qui a tous les droits





# Cas d'usage

---

Forcer l'authentification lors de l'accès au serveur

Restreindre l'accès d'un projet à un groupe d'utilisateurs

Restreindre l'accès au source du projet

Définir qui peut administrer un projet

Définir qui peut administrer le serveur



# Groupes

---

Par défaut, SonarQube définit 3 groupes d'utilisateurs :

- ***anonyme*** : Une personne naviguant sur le site sans authentification  
Par défaut, droit de créer des projets et d'exécuter des analyses si l'authentification n'est pas forcée
- ***sonar-users*** (défaut) : Pas de droit supplémentaire.  
Une personne loggée  
Par défaut, permissions de *anonyme*
- ***sonar-administrator*** : Administrer le système, les portes et les profils qualité



# Utilisateurs et groupes

---

Gestion des utilisateurs :

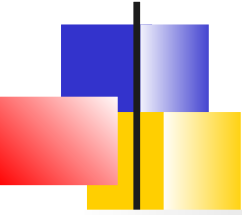
***Settings > Security > Users***

Gestion des groupe

***Administration > Security > Groups***

2 groupes sont prédéfinis :

- ***anyone*** : Tout le monde + l'utilisateur anonyme
- ***sonar-users*** : Tous les utilisateurs de l'annuaire



# Jeton d'accès

---

Chaque utilisateur peut générer un **jeton** qui peut servir à son authentification lors des appels REST au serveur.

Typiquement, lors d'une analyse le jeton peut être fourni via la propriété

***sonar.login***



# Permissions globales

---

***Administration > Security > Global Permissions.***

- ***Administer System***: Administration du serveur
- ***Administer Quality Profiles***: Edition des les profils qualité.
- ***Administer Quality Gates***: Edition des portes qualité
- ***Execute Analysis***: Exécution d'analyse
- ***Create Projects***: Création de projet



# Permissions projet

---

Au niveau du projet :

***Administration -> Permissions***

La visibilité d'un peut être :

- Public
- Privée : Le code source et les mesures ne sont pas accessible par les utilisateurs *anyone*



# Permissions projet

---

Les projets publics et privé peuvent définir les permissions suivantes :

- ***Administer Issues***: Edition des issues
- ***Administer***: Tâches d'administration
- ***Execute Analysis***: Exécution d'analyse

Les projets privé définissent :

- ***Browse***: Accès au projet et édition d'issues
- ***See Source Code***: Accès au code source



# Gabarit

---

Il est possible également de définir des gabarits pour initialiser les permissions par défaut d'un projet.

La création d'un gabarit consiste à fournir :

- un pattern pour la clé du projet
- Un ensemble de permissions





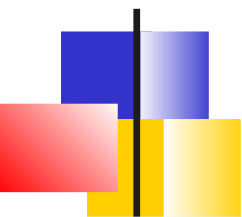
# Historique projet

---

Le **Database Cleaner** est responsable de supprimer certains détails des analyses passées, il peut être configuré.

Par défaut, le cleaner supprime :

- Pour chaque projet :
  - 1 seul snapshot par jour après 1 jour
  - 1 seul snapshot par semaine après 1 mois.
  - 1 seul snapshot par mois après 1 an
  - Seuls les snapshots correspondant à des versions sont conservés après 2 ans.
  - Tous les snapshots vieux de 5 ans sont supprimés
- Tous les issues fermées vieilles de 30 jours sont supprimées
- Historique au niveau d'un package est supprimé



# SonarQube

---

Offre et architecture  
Installation  
Concepts Sonar  
Calculs de métriques  
Administration utilisateurs  
**Workflow**



# SonarQube

---

Le travail quotidien des développeurs et *reviewers* consiste à traiter des *issues*.

Il y en a de 3 types :

- **Code Smell** : Mauvaise pratique de conception rendant difficile la maintenance
- **Bug** : Un problème représentant une erreur dans le code. Cette erreur se manifestera un jour, il faut donc le corriger ASAP
- **Vulnérabilité** : Un problème lié à la sécurité permettant une attaque

Des coûts sont associés au traitement de ces problèmes :

- **Dette technique** : Le temps estimé pour corriger tous les problèmes de maintenabilité
- **Coût de correction** : Le temps estimé pour corriger les bugs et les vulnérabilités



# Sévérité

---

SonarSource définit des sévérités pour les issues :

- **BLOCKER** :  
Haute probabilité d'impacter le comportement en production (fuite de mémoire, de connexion JDBC, ...).  
=> Doit être fixé ASAP
- **CRITICAL** :  
Bug avec faible probabilité d'impact ou défaut de sécurité.  
=> Doit être revu ASAP.
- **MAJOR** :  
Défaut de qualité pouvant impacter la productivité du développeur (code mort, blocs dupliqués paramètres inutilisés, ...)
- **MINOR** :  
Défaut de qualité ayant un léger impact sur la productivité. (Lignes trop longues, "switch" à la place de if, ...)



# Statut des issues

---

Une fois créé, une issue peut prendre les statuts suivants :

- **Ouvert** : Défini par SonarQube sur les nouveaux problèmes
- **Confirmé** : Défini manuellement pour indiquer que le problème est valide
- **Résolu** : Défini manuellement pour indiquer que la prochaine analyse devrait fermer le problème
- **Ré-ouvert** : Défini automatiquement par SonarQube lorsqu'un problème résolu n'a pas été corrigé
- **Fermé** : Défini automatiquement par SonarQube pour les problèmes créés automatiquement.



# Résolution

---

Les issues fermées peuvent avoir l'une des résolutions suivantes :

- **Corrigé/Fixed** : Défini automatiquement lorsqu'une analyse ultérieure montre que le problème a été corrigé ou que le fichier n'est plus disponible (supprimé du projet, exclu ou renommé)
- **Supprimé** : Défini automatiquement lorsque la règle associée n'est plus disponible.

Les issus résolus peuvent avoir l'une des résolutions suivantes :

- **Faux positif** : Défini manuellement
- **Won't fix** : Défini manuellement



# Revue d'issues

---

Le travail du *reviewer* consiste à revoir les résultats de l'analyse et de modifier éventuellement les statuts des problèmes :

- **Confirm** : Reconnaissance du problème. Le statut passe de « Ouvert » à « Confirmé »
- **Fausse détection** : Indique que l'issue détectée par Sonar n'est pas réellement un problème
- **Won't fix** : On reconnaît le problème ; mais on l'accepte en tant que dette technique
- **Changement de sévérité** : Le problème n'a pas la sévérité détectée par Sonar. On change sa sévérité et les prochaines mesures de Sonar tiendront compte du changement
- **Résolu** : Le problème est censé être résolu. Lors de la prochaine analyse, Sonar fermera ou ré-ouvrira l'issue



# Affectation automatique

---

Par défaut, les issues sont affectées au dernier committer de la ligne ou est détectée l'issue si on peut corréler le commiter à un utilisateur SonarQube.

- Elles peuvent cependant être réassignées à quelqu'un d'autre.
- En fonction de ses préférences, le responsable reçoit une notification *email*





# Corrélation d'utilisateur

---

Des corrélations via le login et l'e-mail sont effectuées automatiquement.

Par exemple, si l'utilisateur commette avec son adresse e-mail et que cette adresse e-mail fait partie de son profil SonarQube, les nouveaux problèmes soulevés sur les lignes où l'utilisateur a committé lui seront attribués.

Des corrélations supplémentaires peuvent être effectuées manuellement dans le profil de l'utilisateur



# Édition d'issues

---

Différentes éditions manuelles peuvent également être effectuées sur les issues : commenter, ré-assigner, modifier la sévérité



# Mise en place

---

## **Généralités**

Configuration de l'analyse Sonar  
Personnalisation profils et portes qualité  
Règles personnalisées



# Mener un projet de mesure

---

## L'anti-pattern :

- Prendre un patrimoine qui a déjà du vécu
- Installer tout un tas d'outils de mesure
- Activer « la totale » à sévérité maximale
- Produire des tableaux de chiffres
- Produire un rapport qui constate que tout le patrimoine est uniformément catastrophique
- Et maintenant ?



# Mener un projet de mesure

---

## Introduire la qualimétrie

- Déduire la charte (porte) qualité de l'existant
- Démarrer petit, sur des objectifs clairs
- Mesurer l'évolution
- Étendre petit-à-petit les périmètres de mesure
  - Outils, mesures, patrimoine examiné



# Objectifs de la mesure

---

On peut mesurer pour valider une fourniture externe (Offshore, sous-traitance)

- → Outil *de surveillance*

On peut mesurer en interne

- → plutôt outil *d'assistance*
- Doit être paramétré de façon à *aider* le développeur
- « attention le nom de variable masque un nom d'attribut »
  - « - Ah oui, merci j'avais pas vu »
- « je refuse la livraison car il manque deux lignes de commentaires en moyenne »
  - « - Alors je rajoute deux lignes vides »



# Un indicateur n'est qu'un indicateur

---

La qualimétrie n'est qu'un indicateur de la qualité

Si application trop automatique

- Risque que les développeurs améliorent les indicateurs et non la qualité

Exemple

- Comment augmenter la couverture des tests unitaires ?
- En ajoutant des invocations de méthodes de haut-niveau sans la moindre assertion → intérêt nul en terme de qualité



# Charte de transition

---

Il est intéressant de travailler avec 2 chartes<sup>1</sup> qualité

- Charte stricte qui s'applique aux nouveaux projets
- Charte de transition qui s'applique aux anciens développements
  - Pour les projets en maintenance corrective
  - Elle ne signale que les erreurs sérieuses de fiabilités

Cette charte de transition est une copie de la stricte

- On maintient la sévérité des indicateurs de fiabilité
- On abaisse les indicateurs de maintenabilité
  - Ex : on lève les erreurs de taille, de complexité, de convention de nommage
  - Ex : seuls les nouveaux projets devraient avoir pour cible Java 5 ou + avec utilisation des génériques et des annotations





# Ancien code

---

## Utilisation de la charte de transition sur un ancien code

- On obtiendra ainsi de nombreux avertissements
- Mais on se contentera de ne régler que les problèmes sérieux
- Les avertissements serviront à estimer le coût d'amélioration de la qualité d'un existant vers la qualité stricte d'un nouveau développement

## On peut définir une politique de maintenance

- Ex : toute action de maintenance ne doit dégrader aucun indicateur
- Ainsi, on est obligé de corriger les avertissements sur une classe donnée mais on ne doit pas introduire de problèmes potentiels ni réduire la couvertures des tests unitaires
  - Ce qui impose d'écrire des tests unitaires lors des actions de maintenance, au minimum sur le code ajouté



# Mise en place

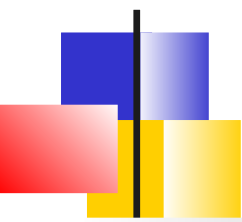
---

Généralités

## **Configuration de l'analyse Sonar**

Personnalisation profils et portes qualité

Règles personnalisées



# Hiérarchie de configuration

---

Les paramètres pour configurer l'analyse peuvent s'effectuer à plusieurs endroits :

- **Paramètres globaux** : Défini via l'interface, ils s'appliquent à tous les projets  
*Administration > Configuration > General Settings*
- **Paramètres projet** : Défini via l'interface, ils surchargent les paramètres globaux. Au niveau projet :  
*Administration > General Settings*
- **Paramètres d'un analyseur** : Défini dans un fichier configuration propre au scanner, surcharge ceux de l'UI
- **Paramètres de ligne de commande** : Ils surchargent tous les autres



# Paramètres obligatoires

---

## Serveur

***sonar.host.url*** L'URL du serveur  
(http://localhost:9000)

## Configuration projet

***sonar.projectKey*** : Un identifiant de projet. Avec Maven : <groupId>:<artifactId>.

***sonar.sources*** : Liste des répertoires contenant les sources séparés par des virgules. Par défaut, l'emplacement Maven



# Paramètres optionnels

---

## Identité projet

***sonar.projectName*** : Nom du projet affiché sur l'UI. Balise `<name>` avec Maven

***sonar.projectVersion*** : La version Balise `<version>` avec Maven.

## Authentification

***sonar.login*** : Le login ou le jeton d'authentification d'un utilisateur avec la permission *Execute Analysis* .

***sonar.password*** : Le mot de passe si on utilise un login

## Service web

***sonar.ws.timeout*** : Timeout pour les appels webservices durant l'analyse



# Configuration projet

---

***sonar.projectDescription*** : Description du projet. Pas d'équivalent Maven

***sonar.links.homepage*** : Page d'accueil du projet. Pas d'équivalent Maven

***sonar.links.ci*** : Lien vers serveur d'intégration. Pas d'équivalent Maven

***sonar.tests*** : Liste des répertoires contenant le source des tests.

***sonar.language*** : Le langage du code source à analyser. Si non spécifié, une analyse multi-langage est effectuée

***sonar.projectBaseDir*** : Si la racine du projet ne correspond au répertoire de démarrage de l'analyse

***sonar.scm.provider*** : Permet d'indiquer explicitement le plugin SCM à utiliser



# Exclusions / Inclusions

---

***sonar.inclusions*** : Liste des gabarits de chemins à inclure dans l'analyse.

***sonar.exclusions*** : Liste des gabarits de chemins à exclure de l'analyse.

***sonar.coverage.exclusions*** : Liste des gabarits de chemins à exclure du calcul de la couverture de code

***sonar.test.exclusions*** : Liste des gabarits de chemins des fichiers de test à exclure de l'analyse.

***sonar.test.inclusions*** : Liste des gabarits de chemins des fichiers de test à inclure dans l'analyse.

***sonar.issue.ignore.allfile*** : Les fichiers contenant cette expression régulière seront ignorés par l'analyse.

***sonar.cpd.exclusions*** : Liste de gabarits de chemins à exclure de la détection de duplication

***sonar.cpd.\${language}.minimumtokens*** : Seuil de détection de duplication en mots. Par défaut : 100

***sonar.cpd.\${language}.minimumLines*** : Seuil de détection de duplication en lignes. Par défaut : 10



# Gabarits

---

Les gabarits présents dans les exclusions/inclusions sont relatifs à la racine du projet.

Ils supportent les caractères suivants :

- \* zéro ou plusieurs caractères
- \*\* zéro ou plusieurs répertoires
- ? un seul caractère

Exemple :

```
sonar.exclusions=**/*Bean.java,**/*DT0.java
```





# Couverture des tests

---

Pour calculer la couverture des tests, SonarQube utilise les rapports d'outils tiers comme *jacoco*

Si l'on sort des configurations par défaut :

- Le plugin peut être précisé via la propriété :

**`sonar.java.codeCoveragePlugin`**

- L'emplacement des rapports générés par

**`sonar.coverage.jacoco.xmlReportPaths`**



# Traces

---

***sonar.log.level*** : Niveau de trace. Par défaut INFO. Plus verbeux DEBUG et TRACE

***sonar.verbose*** : Idem que DEBUG avec en plus les variables d'environnement du client

***sonar.showProfiling*** : Information de profiling. Génère également un fichier *profiler.xml*

***sonar.scanner.dumpToFile*** : Indique un fichier où sont écrites toutes les propriétés passées au scanner



# Mise en place

---

Généralités  
Configuration de l'analyse Sonar  
**Personnalisation profils et portes**  
**qualité**  
Règles personnalisées



# Axes de configuration

---

La configuration d'un projet consiste à spécifier :

- La *Leak Period* : Typiquement la version précédente du projet
- La clé du projet : Permet d'appliquer des permissions par défaut
- Les portes qualité
- Les profils qualité (i.e. les règles actives)
- Les exceptions (exclusions d'analyse, de règles, ...)



# Profils qualité

---

Même si les profils « *Sonar Way* » permettent de démarrer rapidement, il n'est pas recommandé de les utiliser dans les projets car ils ne sont pas éditables et ne peuvent donc pas être personnalisés

=> Il est donc recommandé de créer un nouveau profil et de copier ou hériter du profil « *Sonar Way* » correspondant au langage



# Héritage des profils

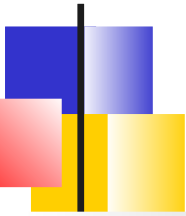
---

Les profils peuvent avoir des relations d'héritage en définissant un profil parent.

Les règles héritées peuvent être surchargées en modifiant le niveau de sévérité

Les mises à jour du profil parent sont répercutés sur les enfants.

- Il peut être intéressant de définir des profils racines pour chaque technologie de l'entreprise
- Hériter du profil *SonarWay* permet de profiter des mises à jour des règles lors d'un *upgrade*.



# Changement dans les règles

---

A tout moment, il est possible de comparer 2 profils et en particulier son profil avec le profil « Sonar way ».

***Page Quality Profiles > Selection du profil 1 > puis Actions > Compare et sélectionner le profil 2 .***

- Cela permet de détecter de nouvelles règles ou les règles dépréciées.

Une recherche en activant le filtre *Available Since* permet également d'isoler les nouvelles règles

Le filtre *Deprecated Rules* isole les règles dépréciées et permet d'accéder aux profils qui les utilise.



# Exclusions de règles

---

En dehors de *SonarWay*, les règles des profils qualité peuvent être activées/désactivées.

SonarQube donne la possibilité de finement configurer ce qu'il va être analysé, en excluant des fichiers/répertoires:

- Complètement
- De la détection de transgression (pour toutes les règles ou certaines règles)
- De la détection de duplications
- Du calcul de la couverture de test





# Ignorer des fichiers

---

4 façons d'ignorer complètement des fichiers :

- Utiliser ***sonar.sources*** pour limiter les fichiers analysés
- Définir les suffixes autorisés  
***Administration > General Settings > [Language]***
- ***source.exclusion*** et ***source.test.exclusion***  
pour exclure des fichiers ou des classes de test
- ***source.inclusion*** et ***source.test.inclusion***  
pour inclure des fichiers ou des classes de test



# Ignorer des issues

---

Il est possible d'ignorer des issues dans certains cas seulement . (Fichiers générés par exemple)

***Administration > General Settings > Analysis Scope > Issues***

- Ignorer des fichiers dont le contenu contient une expression régulière
- Ignorer des blocs de texte. Il faut alors spécifier les marqueurs de début et de fin avec des expressions régulières



# Ignorer avec des critères multiples

---

Ce menu permet de désigner la règle à exclure et les fichiers concernés.

Par ex : *Ignorer la règle "cpp:Union" sur tous les fichiers du répertoire object*

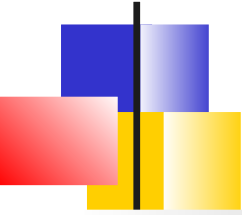
KEY = cpp:Union

PATH = object/\*\*/\*

Cela est également possible par Maven :

```
<sonar.issue.ignore.multicriteria>toto</sonar.issue.ignore.multicriteria>  
<sonar.issue.ignore.multicriteria.toto.resourceKey>object/**/*</  
  sonar.issue.ignore.multicriteria.toto.resourceKey>  
<sonar.issue.ignore.multicriteria.toto.ruleKey>cpp:Union</  
  sonar.issue.ignore.multicriteria.toto.ruleKey>
```

Par l'interface, il est également possible de confirmer les règles à appliquer pour un ensemble de fichiers



# Ignorer les duplications ou de la couverture de test

---

Pour ignorer des fichiers de la détection de duplication :

*Administration > General Settings > Analysis Scope > Duplications*

De la couverture de test :

*Administration > General Settings > Analysis Scope > Code Coverage*



# Journal projet

---

Lorsque SonarQube détecte qu'une analyse a été démarré avec un profil différent, un événement **Quality Profile** est ajouté au journal d'événements du projet.

**Quality Profiles > [ Profile Name ] > Changelog.**

Les utilisateurs avec des privilèges d'administration de profil qualité sont notifiés chaque fois qu'un profil prédéfini est mis à jour.



# Porte qualité

---

La porte qualité est le meilleur moyen pour améliorer la qualité dans un organisation.

Elle répond à la question : Est-ce que cette release peut aller en production ?

Elle est constituée d'un ensemble de conditions booléennes basées sur des seuils de mesure

Les portes qualité doivent être adaptées en fonction des projets.



# Sonar Way

---

Sonar fournit une porte qualité par défaut en mode ***read-only***

Elle s'adresse à stopper l'hémorragie qualité en se basant sur les « Leak periods »

Il faut l'adapter par rapport à sa situation et s'en servir comme objectif à moyen terme



# Condition

---

Chaque condition d'une porte qualité est une combinaison :

- D'une mesure
- D'une période : Value (Date) or Leak  
(Valeur différentielle sur la *Leak period*)
- D'un opérateur de comparaison
- D'une valeur d'avertissement
- D'une valeur d'erreur





# Mise en place

---

Généralités  
Configuration de l'analyse Sonar  
Personnalisation profils et portes qualité  
**Règles personnalisées**



# Création de règles

---

La création de règles personnalisées peut se faire de différentes façons :

- Utiliser les gabarits de règles
- Écrire un plugin Java utilisant l'API de SonarQube
- Ajouter des règles *XPath* en utilisant l'interface Sonar
- Importer des rapports spécifiques à un outil tiers



# Gabarits

---

Créer une nouvelle règle via un gabarit  
peut se faire par l'interface de  
SonarQube

*Rules → Templates Only*

*Create Rule*

Un formulaire dédié est proposé pour  
compléter le gabarit



# Compatibilité

---

Les alternatives de codage de règle sont très dépendantes du langage et du plugin associé.

Pour les plugins fournis de SonarSource :

- L'API Java est disponible pour : COBOL, Java, Javascript, PHP, RPG
- Le code Xpath est disponible pour : Flex, PL/SQL, PL/I, XML
- Outils tiers : PMD, StyleLint, Checkstyle, ESLint, TSLint, ...



# Étapes pour la Java API

---

6 étapes pour mettre en place une nouvelle règle :

- Créer un plugin SonarQube
- Indiquer une dépendance sur le plugin du langage
- Créer les règles voulues
- Générer le plugin SonarQube (fichier jar )
- Placer le fichier jar dans le répertoire  
SONARQUBE\_HOME/extensions/plugins
- Redémarrer le serveur



# Création de projet

---

Sonar fournit des exemples de projet de création de règles pour chaque langage :

<https://github.com/SonarSource/sonar-custom-rules-examples/tree/master/>

Pour Java, un gabarit de projet Maven est disponible

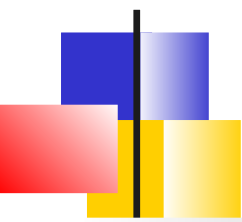


# *pom.xml*

---

En dehors de toutes les dépendances nécessaires, le *pom.xml* définit des propriétés de configuration du plugin

```
<plugin>
  <groupId>org.sonarsource.sonar-packaging-maven-plugin</groupId>
  <artifactId>sonar-packaging-maven-plugin</artifactId>
  <version>1.17</version>
  <extensions>true</extensions>
  <configuration>
    <pluginKey>java-custom</pluginKey>
    <pluginName>Java Custom Rules</pluginName>
    <pluginClass>org.sonar.samples.java.MyJavaRulesPlugin</pluginClass>
    <sonarLintSupported>true</sonarLintSupported>
    <sonarQubeMinVersion>5.6</sonarQubeMinVersion>
  </configuration>
</plugin>
```



# Développement de la règle

---

Au minimum 3 fichiers sont nécessaires lors de la création d'une règle :

- Un fichier de test contenant le code qui servira de données d'entrée pour tester la règle
- Une classe de test contenant les tests unitaires
- Une classe de règle contenant l'implémentation de la règle





# Exemple java

---

Spécification de la règle :

*“Pour une méthode avec un seul paramètre, les types de sa valeur de retour et de son paramètre ne doivent jamais être identique.”*



# Fichier de test

---

```
class MyClass {  
    MyClass(MyClass mc) { }  
  
    int    foo1() { return 0; }  
    void   foo2(int value) { }  
    int    foo3(int value) { return 0; } // Noncompliant  
    Object foo4(int value) { return null; }  
    MyClass foo5(MyClass value) {return null; } // Noncompliant  
  
    int    foo6(int value, String name) { return 0; }  
    int    foo7(int ... values) { return 0;}  
}
```



# Classe de test

---

```
import org.junit.Test;
import org.sonar.java.checks.verifier.JavaCheckVerifier;

@Test
public void test() {
    JavaCheckVerifier.verify("src/test/files/MyFirstCustomCheck.java",
        new MyFirstCustomCheck());
}
```



# Analyseur syntaxique

---

L'analyseur de SonarQube parse un code Java et produit une structure de donnée : l'**arbre de syntaxe**

Chaque construction du langage est représenté avec un arbre de syntaxe **spécifique** et est associé à une **interface** Java qui décrit toutes ses particularités.

Par exemple, une méthode est associé :

- Au type d'arbre  
`org.sonar.plugins.java.api.tree.Tree.Kind.METHOD`
- Et à l'interface  
`org.sonar.plugins.java.api.tree.MethodTree`



# Visiteur

Le design pattern utilisé lors de l'écriture d'une classe de test est le pattern **Visiteur** qui permet de séparer l'algorithme de la logique de parcours de l'arbre syntaxique.

SonarQube appelle lorsque nécessaire des méthodes de type *visitMethod*, *visitAnnotation*, etc. que la règle implémente

Pour limiter l'appel de ces méthodes, la classe de règle peut étendre des types de *SubscriptionVisitor* qui déclare en quoi ils sont intéressé.

- Par exemple : *IssuableSubscriptionVisitor*, *ComplexityVisitor*, ...



# Exemple

```
@Rule(key = "MyFirstCustomCheck",
      name = "Return type and parameter of a method should not be the same",
      description = "For a method having a single parameter, the types of its return value and its
parameter should never be the same.",
      priority = Priority.CRITICAL, tags = {"bug"})
public class MyFirstCustomCheck extends IssuableSubscriptionVisitor {

    @Override
    public List<Kind> nodesToVisit() { // Quels nœuds on veut traiter
        return ImmutableList.of(Kind.METHOD);
    }

    @Override
    public void visitNode(Tree tree) { // Méthode effectuant la vérification
        MethodTree method = (MethodTree) tree;
        if (method.parameters().size() == 1) {
            MethodSymbol symbol = method.symbol();
            Type firstParameterType = symbol.parameterTypes().get(0);
            Type returnType = symbol.returnType().type();
            if (returnType.is(firstParameterType.fullyQualifiedName())) {
                reportIssue(method.simpleName(), "Never do that!");
            }
        }
    }
}
```



# XPath

---

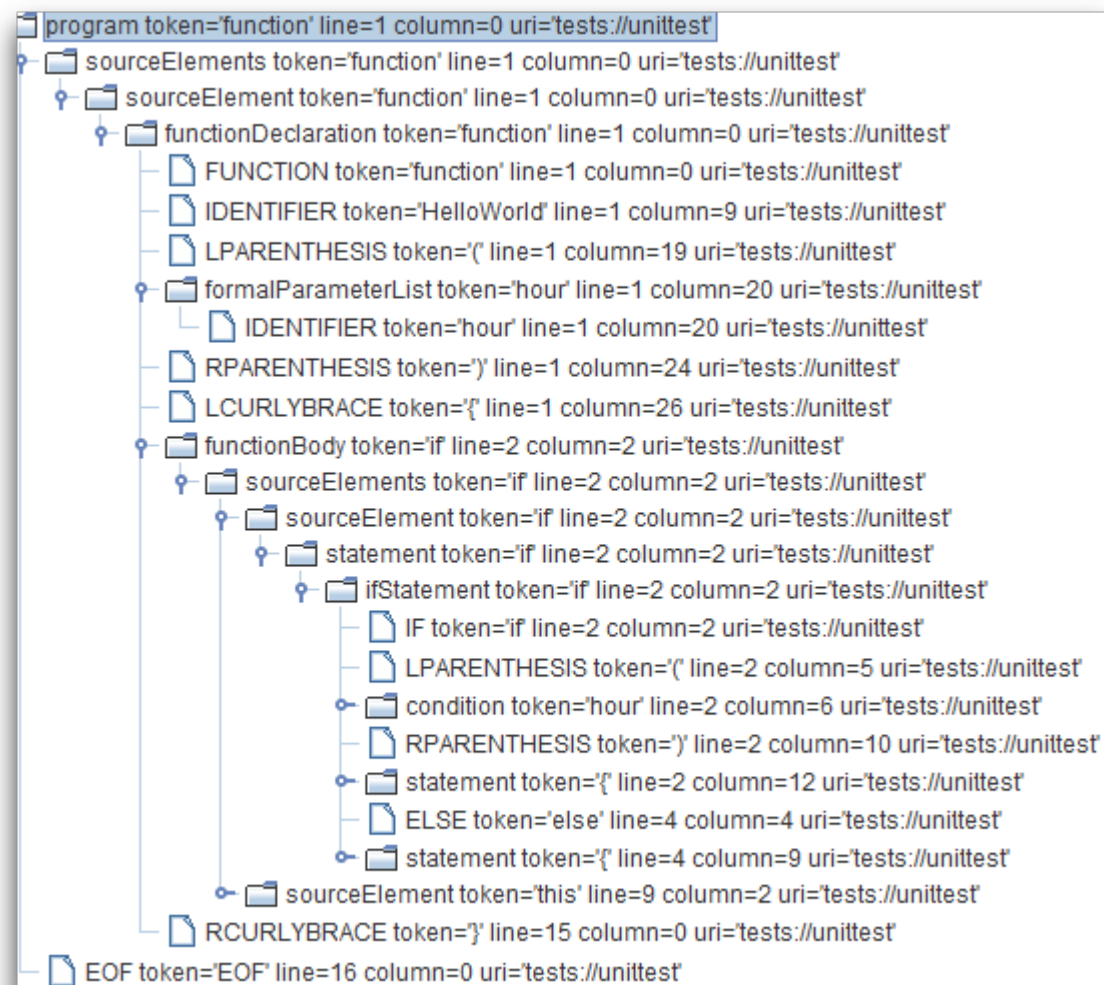
Coder de nouvelles règles en *XPath* est directement possible avec le langage XML

Avec les autres langages supportés, il est nécessaire d'utiliser le SSLR Toolkit propre à son langage.

Le toolkit permet de voir l'arbre syntaxique (Abstract Syntax Tree) du code analysé.

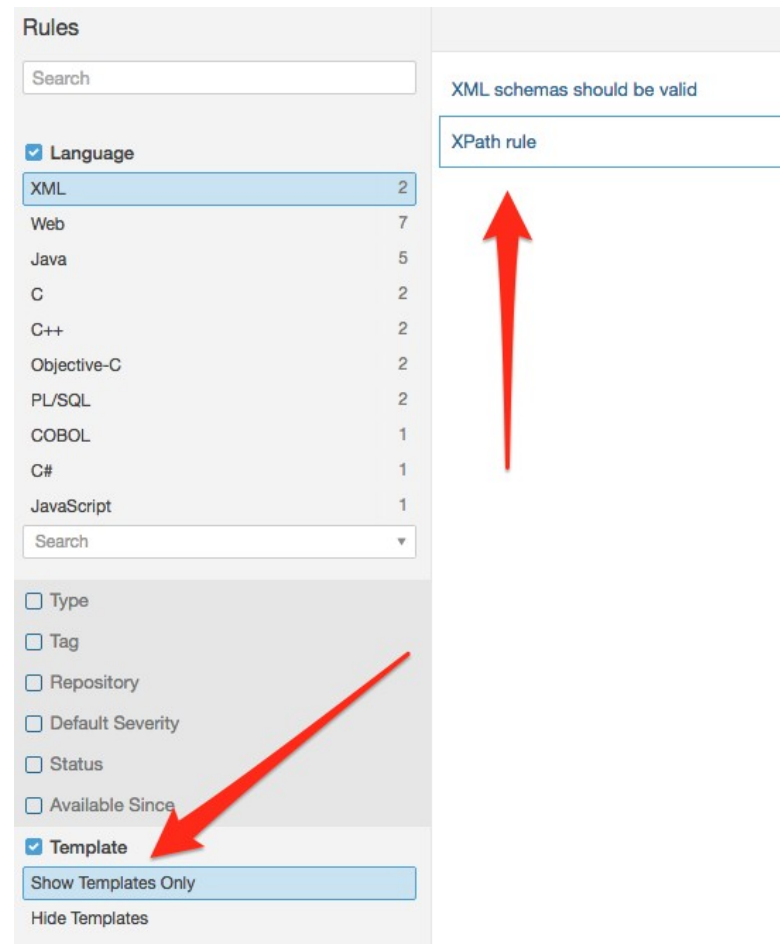
Les expressions XPath sont alors écrites vis à vis de l'arbre.

# Exemple Arbre syntaxique





# Ajout de la règle

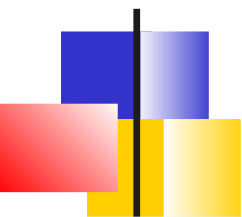


The screenshot shows a web-based configuration interface for rules. On the left, under the 'Rules' header, there is a search bar and a list of languages. The 'Language' checkbox is checked, and 'XML' is selected in the list. Below the language list are several unchecked checkboxes: 'Type', 'Tag', 'Repository', 'Default Severity', 'Status', and 'Available Since'. The 'Template' checkbox is checked, and a red arrow points to it. Below the 'Template' section are buttons for 'Show Templates Only' and 'Hide Templates'. On the right, a preview area shows the rule 'XML schemas should be valid' with an 'XPath rule' button below it. A red arrow points to this button.

Rules	
Search	
<input checked="" type="checkbox"/> Language	
XML	2
Web	7
Java	5
C	2
C++	2
Objective-C	2
PL/SQL	2
COBOL	1
C#	1
JavaScript	1
Search	
<input type="checkbox"/> Type	
<input type="checkbox"/> Tag	
<input type="checkbox"/> Repository	
<input type="checkbox"/> Default Severity	
<input type="checkbox"/> Status	
<input type="checkbox"/> Available Since	
<input checked="" type="checkbox"/> Template	
Show Templates Only	
Hide Templates	

XML schemas should be valid

XPath rule



# SonarLint

---

## **SonarLint**



# SonarLint

---

SonarLint s'intègre dans différents IDEs :  
Eclipse, IntelliJ, VisualStudio, VSCode,  
Atom

<https://www.sonarlint.org/>

Il peut être configurée pour récupérer la  
configuration d'un projet SonarQube.



# Fonctionnalités

---

Analyse temps-réel : Les issues sont indiquées automatiquement, ce mode peut être désactivé.

Mode connecté : *SonarLint* utilise les analyseurs de code, les profils qualité et les configurations définies sur le serveur ou *SonarCloud*

Analyseurs : *SonarLint* supporte les analyseurs *SonarSource* (*SonarJava*, *SonarJS*, ...) ainsi que les règles personnalisées qui étendent les analyseurs. Il ne supporte pas les analyseurs tiers.

Configuration de règle : Si le projet n'est pas connecté, *SonarLint* utilise les règles des profils qualité par défaut.

Exclusion de fichiers et d'issues : Soit dans l'IDE soit sur le serveur



# Administration et intégration

---

## **Intégration SCM**

Intégration via Webhooks

Intégration Jenkins

Monitoring



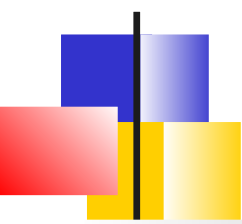
# Introduction

---

L'intégration de svn et git se fait automatiquement.

Cela apporte :

- L'affectation automatique des issues
- L'annotation du code (avec les données de blâme) dans le visualisateur de code
- La détection du nouveau code est pilotée par SCM (Sans SCM, SonarQube détermine le nouveau code à l'aide des dates d'analyse).



# Administration et intégration

---

Intégration SCM

**Intégration via Webhooks**

Intégration Jenkins

Monitoring



# Introduction

---

Webhooks permet d'effectuer des requêtes HTTP POST vers des URLs spécifiées lorsque certains types d'évènements surviennent.

Typiquement, une notification vers d'autres outils lorsque le statut de la porte qualité est disponible.

En général, cela survient durant ou après une analyse mais cela peut également survenir lors de l'édition d'une issue qui ferait changer le statut de la porte qualité





# Requête HTTP

---

La requête HTTP est effectuée :

- Quelque soit le statut de la tâche de fond d'analyse
- Utilise une méthode POST
- A un content type "*application/json*", avec un encoding UTF-8
- Inclut des données sous forme de document JSON

La réponse à la requête doit intervenir sous 10 secondes.



# Example JSON

---

```
{
  "analysedAt": "2016-11-18T10:46:28+0100",
  "project": {
    "key": "org.sonarqube:example",
    "name": "Example"
  },
  "properties": {
  },
  "qualityGate": {
    "conditions": [
      {
        "errorThreshold": "1",
        "metric": "new_security_rating",
        "onLeakPeriod": true,
        "operator": "GREATER_THAN",
        "status": "OK",
        "value": "1"
      },
      {
        "errorThreshold": "1",
        "metric": "new_reliability_rating",
        "onLeakPeriod": true,
        "operator": "GREATER_THAN",
        "status": "OK",
        "value": "1"
      }
    ],
    ...
  }
}
```



# Example JSON (2)

---

```
{
  {
    "errorThreshold": "1",
    "metric": "new_maintainability_rating",
    "onLeakPeriod": true,
    "operator": "GREATER_THAN",
    "status": "OK",
    "value": "1"
  },
  {
    "errorThreshold": "80",
    "metric": "new_coverage",
    "onLeakPeriod": true,
    "operator": "LESS_THAN",
    "status": "NO_VALUE"
  }
],
"name": "SonarQube way",
"status": "OK"
},
"serverUrl": "http://localhost:9000",
"status": "SUCCESS",
"taskId": "AVh21JS2JepAEhwQ-b3u"
}
```



# Administration et intégration

---

Intégration SCM  
Intégration via Webhooks  
**Intégration CI**  
Monitoring



# Introduction

---

L'intégration avec la plateforme de CI/CD consiste à faire échouer la pipeline lorsque la porte qualité n'est pas passée

Sonarqube offre du support pour :

- Jenkins
- Github
- Bitbucket



# Intégration Jenkins

---

Le plugin ***SonarQube Scanner*** permet de centraliser la configuration de SonarQube dans Jenkins

L'analyse d'un projet peut alors être définie comme étape d'un build

Une fois l'analyse terminée, un statut de qualité est remonté sur l'UI Jenkins et un lien permet d'accéder aux tableaux de bord Sonar



# Configuration

---

La mise en place consiste à :

- Définir l' ou les instances de SonarQube

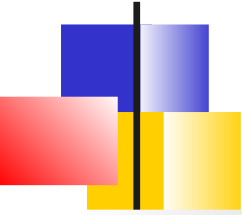
*Manage Jenkins → Configure System → SonarQube configuration → Add SonarQube*

- Définir les scanners à utiliser (Exemple Maven)

*Manage Jenkins → Configure Tools → SonarQube scanner*

- Configurer un projet pour utiliser le scanner

# Job Legacy



## Lancer une analyse avec SonarQube Scanner

Tâche à lancer

JDK

Le JDK à utiliser pour cette analyse SonarQube

Chemin vers les propriétés du projet

Propriétés de l'analyse

Additional arguments

Options de la JVM





# Pipeline plugin

---

Le plugin ***SonarQubeScanner*** propose  
2 steps :

- ***withSonarQubeEnv*** : Prépare l'environnement de l'agent pour l'exécution de Sonar
- ***waitForQualityGate*** : Pause la pipeline en attendant que l'analyse soit terminée, et retourne le statut de la porte qualité



# Example

---

```
node {  
  stage('SCM') {  
    git 'https://github.com/foo/bar.git'  
  }  
  stage('SonarQube analysis') {  
    // requires SonarQube Scanner 2.8+  
    def scannerHome = tool 'SonarQube Scanner 2.8';  
    withSonarQubeEnv('My SonarQube Server') {  
      sh "${scannerHome}/bin/sonar-scanner"  
    }  
  }  
}
```



# Exemple Maven + timeout

---

```
node {
    stage('SCM') {
        git 'https://github.com/foo/bar.git'
    }
    stage('SonarQube analysis') {
        withSonarQubeEnv('My SonarQube Server') {
            sh 'mvn clean package sonar:sonar'
        }
    }
}

// Pas besoin d'occuper un agent
stage("Quality Gate"){
    timeout(time: 1, unit: 'HOURS') {
        def qg = waitForQualityGate() // Réutilisation de taskId setté par
        withSonarQubeEnv
        if (qg.status != 'OK') {
            error "Pipeline aborted due to quality gate failure: ${qg.status}"
        }
    }
}
```



# Administration et intégration

---

Intégration SCM  
Intégration via Webhooks  
Intégration Jenkins  
**Monitoring**



# Introduction

---

Le premier point de surveillance est l'API Web :

***<http://<server>/api/system/health>***

Elle donne un premier diagnostic du serveur

Les autres points à surveiller sont :

- La mémoire du processus Java
- Les Beans JMX du processus Java
- L'activité Elasticsearch



# Processus java

---

Une instance de SonarQube consiste en l'exécution de 3 process Java :

- Le serveur Web
- Le moteur d'analyse
- ElasticSearch

Leur configuration mémoire peut s'effectuer dans `conf/sonar.properties` :

- `sonar.web.javaOpts`
- `sonar.ce.javaOpts`
- `sonar.search.javaOpts`



# Mbeans JMX

---

En plus des Mbeans standards, SonarQube ajoute des Mbeans spécifiques à chaque composant de l'architecture :

- ComputeEngine
- Database
- ElasticSearch
- SonarQube

Tous ces Mbeans exposent des attributs read-only, il n'est pas possible de modifier ou réinitialiser en cours d'exécution



# Attributs de ComputeEngine MBean

---

***ProcessingTime*** : Temps passé en ms pour exécuter les tâches de fond (analyse ou autre)

***ErrorCount*** : Nombre de tâche de fond ayant échoué depuis le dernier redémarrage

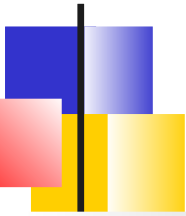
***PendingCount*** : Nombre de tâches de fond en attente

***InProgressCount*** : Nombre de tâches de fond en exécution

***SuccessCount*** : Nombre de tâches de fond ayant réussi depuis le dernier redémarrage

***WorkerCount*** Number : Nombre de worker simultanés





# Attributs du Mbean Database

Ce Mbean est présent pour les processus ComputeEngine et WebServer

***MigrationStatus*** : Soit UP\_TO\_DATE, REQUIRES\_UPGRADE, REQUIRES\_DOWNGRADE, FRESH\_INSTALL

***PoolActiveConnections*** : Nombre de connexions actives dans le pool

***PoolIdleConnections*** : Nombre de connexions en attente dans le pool

***PoolInitialSize*** : Taille initiale du pool

***PoolMaxActiveConnections*** : Taille max du pool

***PoolMaxIdleConnections*** : Taille max de connexions idle

***PoolMaxWaitMillis*** : Valeur d'attente max dans le pool

***PoolRemoveAbandoned*** : true, false

***PoolRemoveAbandonedTimeoutSeconds*** : En secondes



# Attributs du Mbean ElasticSearch

---

***NumberOfNodes*** : Nombre de nœuds  
Esearch dans SonarQube.

***State*** : Statut du cluster : GREEN,  
YELLOW, RED

ElasticSearch propose également une  
API de monitoring très complète. Point  
d'entrée :

*[http://<server>/\\_cluster/health](http://<server>/_cluster/health)*



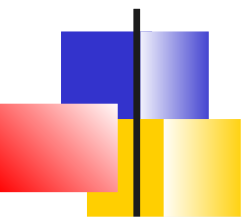
# Attributs du MBean de SonarQube

---

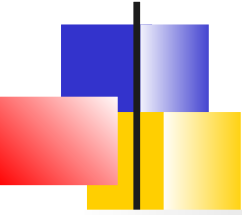
***LogLevel*** : Niveau de trace : INFO, DEBUG, TRACE

***ServerId*** : ID du serveur

***Version*** : Version de SonarQube



# Annexes



# Functionality (Fonctionnalité)

---

Capacité à délivrer les fonctionnalités attendues

- Suitability (Pertinence)
  - Capacité à délivrer les fonctions appropriées aux tâches/objectifs de l'utilisateur
- Accuracy (Exactitude)
  - Capacité à fournir les résultats attendus avec la précision attendue
- Interopability (Interopérabilité)
  - Capacité à interagir avec les systèmes spécifiés
- Security (Sécurité)
  - Capacité à protéger les données contre les personnes/systèmes non autorisées
- Functionality compliance (Conformité fonctionnelle)
  - Capacité à respecter les standards/conventions/lois relatives aux fonctionnalités



# Reliability (Fiabilité)

---

Capacité à maintenir le niveau de performance attendu dans les conditions d'utilisation spécifiées

- Maturity (Maturité)
  - Capacité à éviter les plantages en réponses aux erreurs internes
- Fault tolerance (Tolérance aux pannes)
  - Capacité à maintenir le niveau de performance attendue en cas de défauts ou de mauvaise utilisation de ses interfaces
- Recoverability (Capacité de récupération)
  - Capacité à rétablir le niveau de performance attendu et à restaurer les données défectueuses après un plantage
- Availability (Disponibilité)
  - Capacité à être prêt à délivrer les fonctionnalités attendues
- Reliability compliance (Conformité de fiabilité)
  - Capacité à respecter les standards/conventions/lois relatives à la fiabilité

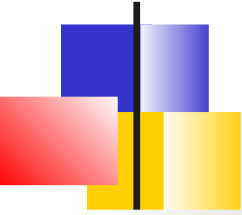


# Usability (Utilisabilité)

---

Capacité à être compris, appris, utilisé, convivial pour l'utilisateur dans les conditions d'utilisation spécifiées

- Understandability (Intelligibilité)
  - Capacité à être compris par l'utilisateur pour réaliser une tâche particulière
- Learnability (Simplicité d'apprentissage)
  - Capacité à permettre à l'utilisateur d'apprendre son utilisation
- Operability (Opérabilité)
  - Capacité à permettre à l'utilisateur de faire fonctionner le logiciel et à le contrôler
- Attractiveness (Attrait)
  - Capacité à attirer/plaire à l'utilisateur
- Usability compliance (Conformité d'usage)
  - Capacité à respecter les standards/conventions/guides relatifs à l'utilisabilité



# Efficiency (Efficacité)

---

Capacité à fournir les performances appropriées selon les ressources utilisées dans les conditions spécifiées

- Time behaviour (Temps de réponse)
  - Capacité à fournir les réponses appropriées dans les temps et aux débits prévus
- Resource utilisation (Utilisation des ressources)
  - Capacité à utiliser les quantités et types de ressources appropriés
- Efficiency compliance (Conformité d'efficacité)
  - Capacité à respecter les standards/conventions relatifs à l'efficacité



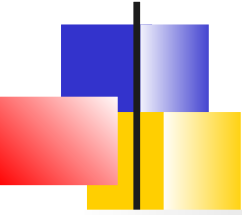


# Maintainability (Maintenabilité)

---

Capacité à être modifié (corrigé, amélioré, adapté) selon les changements (environnement/spécification)

- Analysability (Analysabilité)
  - Capacité à diagnostiquer les déficiences et causes d'erreurs et à trouver les parties à modifier
- Changeability (Changeabilité, adaptabilité, modifiabilité)
  - Capacité à implémenter des modifications de spécifications
- Stability (Stabilité)
  - Capacité à éviter des effets non attendus après des modification
- Testability (Testabilité)
  - Capacité à être validé, en particulier en cas de modification
- Maintainability compliance (Conformité de maintenabilité)
  - Capacité à respecter les standards/conventions relatifs à la maintenabilité



# Portability (Portabilité)

---

Capacité à être transféré d'un environnement à un autre

- *Adaptability* (Adaptabilité)
  - Capacité à être adapté aux environnements spécifiés
- *Installability* (Facilité d'installation)
  - Capacité à être installé dans un environnement spécifié
- *Co-existence* (Coexistence)
  - Capacité à cohabiter/partager des ressources avec d'autres logiciels
- *Replaceability* (Facilité de remplacement)
  - Capacité à être remplacé par un logiciel spécifié équivalent (ex : upgrade)
- *Portability compliance* (Conformité de portabilité)
  - Capacité à respecter les standards/conventions relatifs à la portabilité



# Qualité d'usage

---

Beaucoup plus simple : 4 critères

- *Effectiveness* (Efficacité)
  - Capacité à permettre à l'utilisateur d'achever ses buts dans le contexte spécifié
- *Productivity* (Productivité)
  - Capacité à permettre à l'utilisateur d'utiliser les ressources appropriées pour achever ses buts
- *Safety* (Sécurité)
  - Capacité à maintenir un niveau de sécurité acceptable
- *Satisfaction* (Satisfaction)
  - Capacité à satisfaire l'utilisateur dans les conditions d'utilisation prévues

# Métriques définies par la norme

Chacun des critères est accompagné de métriques

- Définition précise (comment mesurer, interpréter, type d'échelle,...)

External suitability metrics							
Metric name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Source input to measurement
Functional adequacy	How adequate are the evaluated functions?	Number of functions that are suitable for performing the specified tasks comparing to the number of function evaluated.	$X=1-A/B$ A= Number of functions in which problems are detected in evaluation B= Number of functions evaluated	$0 \leq X \leq 1$ The closer to 1.0, the more adequate.	Absolute	$X = \text{Count} / \text{Count}$ A= Count B= Count	Requirement specification (Req. Spec.) Evaluation report
Functional implementation completeness	How complete is the implementation according to requirement specifications?	Do functional tests (black box test) of the system according to the requirement specifications. Count the number of missing functions detected in evaluation and compare with the number of function described in the requirement specifications.	$X = 1 - A / B$ A = Number of missing functions detected in evaluation B = Number of functions described in requirement specifications	$0 \leq X \leq 1$ The closer to 1.0 is the better.	Absolute	$A = \text{Count}$ $B = \text{Count}$ $X = \text{Count} / \text{Count}$	Requirement specification Evaluation report